# SCO® UNIX® System V Development System

## Release and Installation Notes

Release 3.2.2

The Santa Cruz Operation™

Document Version: 3.2.2C
Date: 18 July 1990

# SCO UNIX Development System Release and Installation Notes Release 3.2.2

---

SCO UNIX Development System
Release and Installation Notes
18 July 1990

# 1. Preface

This document contains important information about the SCO UNIX Development System.

These notes are divided into two parts: software notes, and Appendix A, "New and Revised Manual Pages." The appendix is not listed in the table of contents.

The software notes are organized into the following sections:

- Installation Notes
- General Software Notes
- C Language Notes
- SCO UNIX Development System Documentation Notes
- POSIX Implementation Defined Issues
- SCO 3.2 Product Engineering Toolkit Notes
- ISAM Development System Notes

We are always pleased to hear of users' experiences with our products, and welcome recommendations on how they can be made even more useful. All written suggestions are given serious consideration.

## 1.1 Conventions Used in These Notes

Utilities and commands are printed in **boldface** type, with the reference guide section following in parentheses. For example, **fsck**(ADM) refers to the **fsck** manual page in the System Administration (ADM) section. Filenames are in *italics*. For example, you see the */etc/default/filesys* file.

## 1.2 Contents of the Distribution

The components of your Development System package include:

- SCO UNIX System V Development System

  The Development System provides programs and commands for creating, debugging, and maintaining C language and assembly language programs. It includes:

  - **CodeView**, a window-oriented debugger

  - **MASM**, the Microsoft® Macro Assembler, which lets you link, assemble, and run assembly language programs.

- SCO CGI Development System

  SCO CGI (Computer Graphics Interface) provides the tools to create graphics applications packages for the System V environment. The SCO CGI package also contains selected graphics device drivers for developers to test their graphics applications on various peripherals.

  SCO CGI is a separately installable product, and installation instructions are included in the *CGI Release and Installation Notes*.

- SCO Product Engineering Toolkit

  The Product Engineering Toolkit allows you to create a standard installation interface for your applications. The interface is consistent with that of SCO products, and provides users with all of the options of the **custom**(ADM) utility.

- SCO ISAM

  SCO ISAM (indexed sequential access method) provides a complete set of tools for a programmer to build custom

data management procedures into any type of application.

### 1.2.1 Documentation

For a complete list and description of the accompanying documentation, see the documentation roadmap included in the Development System package.

### 1.2.2 Installation Media

The SCO UNIX Development System is distributed on 96 tpi and 135 tpi diskettes for standard architecture machines. The software is grouped into several functional areas called *packages*. This makes customizing your system easier, because you can use the **custom**(ADM) utility to add or delete groups of related programs.

### 1.2.3 Serialization Card

This contains your serial number and activation key, both of which are needed to install the SCO UNIX Development System.

## 1.3 Software Support

Software support is available to customers who purchased the SCO UNIX Development System for use in the United States and Canada. If you purchased it for use outside of the U.S. or Canada, contact your distributor or retailer for support information.

Software support is described on the Customer Registration Form included with your SCO UNIX Development System documentation. Complete the form and return it to us within five days of receiving your software package. To be eligible for our support services, please supply all information requested on the form.

# 2. Installation Notes

You must have SCO UNIX System V/386 Release 3.2 installed on your computer to use the SCO UNIX System V/386 Development System Release 3.2.

## 2.1 Removing an Earlier Release

If you currently have an earlier release of the SCO UNIX System V Development System installed on your system, you must remove it according to the following procedure before you install the current release:

1. Save a copy of */etc/perms/dsmd* in another directory. For example, type:

   **cp /etc/perms/dsmd /tmp/foo/dsmd**

   Removal of the SCO UNIX System V Development System removes this file as well, and it is required for subsequent reinstallation of the Development System. Do not store the file directly under the */tmp* directory, as it can be removed from there as well.

2. Using **custom**(ADM), remove the previous version of the SCO UNIX System V Development System.

3. Exit **custom**, then restore */etc/perms/dsmd* from the copy you made in step 1.

## 2.2 Installing the SCO UNIX Development System

Before you install the Development System, make sure that you have:

- the Development System installation media
- the Operating System "N" volume diskettes. Depending on your installation, you are prompted to insert one or more of these diskettes as part of your Development System installation procedure.
- your Development System serial number and activation key

To install the Development System, follow these steps:

1. Log in as *root* (the super user) and bring the system to system maintenance mode.

2. Enter the following command at the prompt:

   **custom**

3. The main **custom** menu is displayed. Select Install.

4. Select A New Product.

5. If you want to install the entire Development System, select Entire Product. If you want to install specific packages only, select Packages.

6. This message is displayed:

   ```
   Insert: Distribution
           Floppy Volume 1
   ```

   Insert the volume labeled "D01" into floppy drive 0, and press ⟨Return⟩.

7. If you chose the Entire Product option, skip to step 8.

   If you chose to install Packages, the package list is displayed:

   ```
   ALL     Entire Development System set
   PERM    Development System permlists
   SOFT    Basic Software Development Tools
   XNXDEV  XENIX cross development include files and utils
   386XDEV XENIX 386 Cross Development libraries
   286XDEV XENIX 286 cross development libraries
   LEX     Generates programs for lexical analysis
   YACC    Yet another compiler-compiler
   CFLOW   Generates C flow graphs
   LINT    Syntax and usage check files and tools
   SCCS    Source code control system
   DOSDEV  DOS cross development utilities and libraries
   ```

```
OS2DEV  OS/2 cross development libraries and utilities
MESGCAT XPG Message Catalog Utils
SAMPLE  Sample Shell Scripts for the PE Toolkit
MAN     Development system manual pages
PETKIT  The Distribution Making Tools
KITMAN  The Man Package for the PE Toolkit
ISOMF   SCO ISAM OMF Format Libraries
ISCOFF  SCO ISAM COFF Format Libraries
```

Select the packages you want to install by moving the highlight to each desired package and pressing ⟨Space⟩. Selected packages are marked with an asterisk (*). When you have marked all the packages that you want to install, press ⟨Return⟩.

8. This message is displayed:

```
Insert: Software Development System
        Floppy Volume D1
```

Depending on which packages you chose to install, a different volume number might be requested first.

Insert the requested volume and press ⟨Return⟩.

9. Insert the other volumes as they are requested. When you have installed the last package, the Restricted Rights Legend is displayed, followed by this message:

```
Product Serialization

When prompted, use the serial number and activation
key included with the Product distribution.

Enter your serial number or enter q to quit:
```

10. Type your serial number, and press ⟨Return⟩.

11. Now this message is displayed:

```
Enter your activation key or enter q to quit:
```

Type your activation key, and press ⟨Return⟩.

12. After some time, this message is displayed:

    ```
    press any key to continue.
    ```

    Press any key. You are prompted to insert one or more of the "N" volumes. The "N" diskettes are not part of the Development System distribution (although **custom** might refer to them as "Development System" diskettes). They are volumes from the operating system distribution and certain operating system dependent files and utilities must be extracted from them when you install the Development System.

13. After the "N" volumes are installed, you return to the **custom** menu. The installation is now complete. Select Quit from the **custom** menu to exit **custom**.

After you install the SCO UNIX Development System, check the file */etc/default/cc* to confirm that the default settings for the compiler meet your needs.

## 3. Development System Software Notes

### 3.1 ANSI Conformance

The Microsoft C compiler and libraries included with the SCO UNIX System V/386 Development System are substantially conformant with the draft proposed ANSI C Programming Language standard X3.159-198X dated May 13, 1988.

The Development System manual pages in the *Programmer's Reference* include a section on standards conformance to provide a guide to those functions that are compliant with ANSI C, IEEE and FIPS POSIX, and X/Open or SVID.

### 3.2 CodeView

The CodeView debugger shipped with this release recognizes the following options:

| | |
|---|---|
| **-c** | special commands |
| **-t** | sequential mode |
| **-b** | monochrome display |

See the *CodeView Debugger* manual for more information on CodeView options.

In the CodeView documentation, there are special functions described that are accessed using the ⟨Alt⟩ key in conjunction with other keys. By default, these functions are not available. The file */usr/lib/keyboard/cv*, which is included in your standard distribution, must be renamed to replace the file */usr/lib/keyboard/keys* in order to access the special functions of keys used in conjunction with the ⟨Alt⟩ key. The original */usr/lib/keyboard/keys* file should be saved as a precaution. For example, to perform the above described operation, log in as *root* (the super user) and give the following commands:

**mv /usr/lib/keyboard/keys /usr/lib/keyboard/keys.bkp**
**cp /usr/lib/keyboard/cv /usr/lib/keyboard/keys**

You should also add the following command to the */etc/rc* file to direct your system to customize the keyboard automatically at boot time:

**mapkey**

However, if your system has an ⟨F12⟩ key, the ⟨Alt⟩ key functions can be used by first pressing ⟨F12⟩ and then the ⟨Alt⟩⟨*key*⟩ combination needed.

Floating-point support is not yet implemented in CodeView.

CodeView cannot be used to debug 286 binaries.

Known problems with the CodeView software include:

- Mouse support in CodeView is not available in this release.

- The shell escape does not function correctly at this time.
- The E command (Execute until keypress) does not work properly.
- Regular expression searching is implemented using the *regex* library, rather than the notation described in the *CodeView Debugger* documentation.
- Programs that interact with the video driver to map the video memory, to use graphics mode, or to switch screens cause CodeView to behave irregularly. However, use of the normal multiscreens is available without difficulty.

### 3.3 crypt(C) and crypt(S) Libraries

The **crypt**(C) utility and **crypt**(S) libraries are not included in this release. If you are a United States resident, you can request a copy of **crypt** by calling your support center.

### 3.4 cxref(CP)

**cxref**(CP) does not correctly handle function prototypes. Its syntax analysis is based on an older version of the compiler. To work around this problem, make function prototypes subject to a conditional compilation definition. For example:

```
#ifndef NO_PROTOTYPE
int func( char *, long );
#endif
```

Then use this flag when using **cxref**:

**-DNO_PROTOTYPE**

### 3.5 help(CP)

The **help**(CP) package provides explanations of SCCS error messages only. For information on other commands, use **man**(CT).

### 3.6 ld(CP)

The **-r** option of **ld**(CP) does not work correctly for object files that contain the additional symbol table information used by the symbolic debugger. If you use the **-r** option of **ld** to partially link together several object modules that contain symbolic debugging information, the code, data, and relocation information in the resulting object module is correct, but the symbolic debugging information is unusable.

### 3.7 ranlib(CP)

When you create or modify an OMF library, you must process it with **ranlib**(CP) before you use it. If you do not, **ld**(CP) fails.

Do not process COFF libraries with **ranlib**.

### 3.8 nm(CP)

The **-n** option of **nm**(CP) generates an error with long name lists. To sort a long name list, instead of using the **-n** option, use the command line:

**nm** *executable* | **sort**

### 3.9 OMF/COFF Utilities

The following utilities determine whether a subject file is of OMF or COFF format and execute different binaries in each case:

**ld**
**ar**
**nm**
**size**
**strip**

Therefore, changing or moving any file that these utilities depend upon (the files in */lib/coff* and */lib/xout*) causes an error. You see the message:

`can't exec <`*filename*`>`

### 3.10 sccs(CP) Line Size

There is a limit of 508 characters on any physical line of any file placed under **sccs**. Writing a changed file with a line containing more than 508 characters results in corruption of the file.

### 3.11 sdb(CP)

Programs that are debugged with **sdb** must be compiled and linked with the following options:

**cc -Zi** or **cc -g**

**ld -g**

The following features of **sdb** that are described in the documentation are not supported in this release:

- **sdb** is not available for OMF binaries.
- Pattern matching for function and variable names is not available. For example, the following string should display the values of all variables with names beginning with "x":

  x*/

- The "." command to redisplay the value of the last variable typed is not available.
- The command to read **sdb** commands in from a file works correctly only when there is no space placed between the "<" character and the filename:

  * <*file*

These features are available with CodeView.

## 3.12 Converting Binaries

The binary conversion utilities **cvtcoff**(M) and **cvtomf**(M) can be used to convert compiled binaries between Common Object File Format (COFF) and OMF XENIX® file format (OMF), with some restrictions.

Any source code using the following files or the related system calls or library routines should not be converted after compilation of any kind:

| File | Description |
|---|---|
| *a.out.h* | COFF vs. *x.out* binary formats |
| *core.h* | core file analysis |
| *lockcmn.h* | file lock requests for *x.out* binaries |
| *mon.h* | **monitor**(S) and profiling data, such as *mon.out* |
| *tinfo.h* | terminfo curses |
| *sys/fcntl.h* | **fcntl**(S) and **lockf**(S) file lock requests |
| *sys/locking.h* | **locking**(S) system call |
| *sys/shm.h* | shared memory segments (IPC) |
| *sys/signal.h* | SIGPOLL value |
| *sys/utsname.h* | **uname**(S) system call |

### 3.12.1 Using cvtomf(M)

The **cvtomf**(M) utility translates OMF relocatable object files to COFF relocatable object files. It cannot convert *x.out* format executables.

---

*Important*

**cvtomf**(M) was designed to be used by the Microsoft C compiler driver. Using this utility in any other manner can introduce binary incompatibilities in the resulting COFF file that adversely affect runtime behavior.

---

The usage listed in the **cvtomf**(M) manual page is not complete. Here is the correct usage:

**cvtomf** [**-g**] [**-o** *outfile*] *filelist*

The **-g** option causes symbolic debug data to be converted. Symbolic data does not translate properly if the OMF file contains multiple modules, produced by incremental linkage using **ld**(M) **-r**. This option functions the same as the **-g** option to **cc**(CP).

The **-o** option allows you to specify an output filename when a single file is being converted. By default, the COFF output overwrites the OMF input.

### 3.12.2 Using cvtcoff(M)

The **cvtcoff**(M) utility translates a COFF relocatable object file, archive, or executable to the corresponding OMF/*x.out* format file.

---

*Important*

**cvtcoff**(M) was designed to be used by the Microsoft linker and CodeView. Using this utility in any other manner can introduce binary incompatibilities in the resulting OMF/*x.out* file that adversely affect runtime behavior.

---

The usage listed in the **cvtcoff**(M) manual page is not complete. Here is the correct usage:

**cvtcoff** [-**gv**] [-**o** *outfile*] *file*

The **-g** option causes symbolic debug data to be converted. This option can be used in the conversion of COFF executables only, and not with relocatable objects or archives. This option functions the same as the **-g** option to **cc**(CP).

By default, the output file is named *x.out*. The **-o** option lets you specify a different filename.

### 3.13 XENIX Software Products

When installing XENIX products on your SCO UNIX Operating System, note that the libraries and include files provided are for producing *x.out* format binaries only. Therefore, until other products are updated for 3.2, you should use the **-xout**, **-x2.3**, or the **-xenix** options when compiling using older XENIX products.

## 4. C Language Notes

### 4.1 brkctl(S) Library

SCO UNIX System V does not support 386 processes that have more than one data segment. For compatibility, a special library (*/lib/386/Slibbrkctl.a*) is provided that maps **brkctl**( ) functions into calls to **sbrk**(S). This provides for the most common uses of **brkctl**(S), such as the allocation of additional memory.

Programs that rely on allocating multiple data segments and manipulating the sizes of those segments need to be altered to work under SCO UNIX System V.

The following describes the functionality implemented by the 386 *libbrkctl.a*. Note in particular that the 386 **brkctl**(S) returns a near pointer and that the third argument is also a near pointer. If you do not wish to alter your source code when compiling for the 386, you should include **-Dfar=** on the **cc**(CP) command line. This causes the preprocessor to remove all "far" keywords from your code.

```
#include <sys/brk.h>
char *brkctl(cmd, inc, ptr)
int cmd;
long inc;
char *ptr;
```

When called from 386 processes, **brkctl**( ) does not cause the allocation or de-allocation of far data segments. It can only be used to increase or decrease the memory allocation in the single data segment available to 386 processes.

Any of the commands **BR_IMPSEG**, **BR_ARGSEG** or **BR_NEWSEG** can be used as they all have the same effect.

The *ptr* argument in the above program example is ignored.

It is unusual to allow the use of **BR_NEWSEG** with 386 processes because it is not possible to allocate additional data segments. However, **BR_NEWSEG** is commonly used in small and middle model 86/286 processes to allocate additional memory. Therefore, it was decided to give **BR_NEWSEG** the same functionality as **BR_IMPSEG**.

In order to link a 386 binary with this version of **brkctl**( ) you should specify **-lbrkctl** on the **cc**(CP) command line.

### 4.2 cc(CP) Defaults

The default code generation model for the 80386 is **-M3e**. (80386 with non-ANSI extensions enabled.) This default can be changed by editing */etc/default/cc*. See the **cc**(CP) manual page for details.

### 4.3 International Development

This section describes enhancements provided by the SCO International Development System.

### 4.3.1 conv(S) Routines

Two macros, **todigit** and **toint**, have been added. The **todigit** macro returns the digit character associated with the integers 0-9. The **toint** macro does the opposite.

The macro definitions of **toupper** and **tolower** have been replaced by library functions. This is to introduce behavior that is a superset of the ANSI standard and equivalent to SVID and X/OPEN.

The arguments to **toupper** and **tolower** are converted to integers. For this reason, care should be taken that character arguments are supplied as unsigned characters to avoid problems with sign-extension of 8-bit character values.

### 4.3.2 ctype(S) Routines

Several macros have been revised to accommodate 8-bit characters. While **isascii** is defined on all integer values, the rest are defined on integers, the value of which can be represented as an unsigned char, or is equal to the single non-character value EOF.

### 4.3.3 Limitations

Although the underlying libraries used by the Development System and other utilities have been modified to function with 8-bit characters, only a key number of utilities have been verified to function properly in all cases. Unless the utility or routine is listed as verified in these notes, data files and keyboard input should be restricted to 7-bit ASCII characters. For example, in the case of the C compiler, 8-bit characters can be used in character and string literals. However, for reasons of compatibility with other utilities and C compilers on other systems, it is important to confine use of 8-bit characters to data files and to use *\nnn* escapes to specify 8-bit characters within C source.

### 4.3.4 Native Language Support

A number of routines have been introduced to ensure compatibility with the *X/OPEN Portability Guide*. The routines are **nl_init**, **nl_cxtime**, **nl_printf**, **nl_langinfo**, **nl_scanf**, and **nl_strcmp**.

The **nl_init** routine is the X/OPEN equivalent of **setlocale**.

### 4.3.5 Regular Expression Routines

The **regexp** and **regex** routines have been verified for 8-bit operation. The **regexp** syntax has been extended to accommodate 8-bit characters. Refer to the **ed**(C) manual page for more information.

As part of the modifications for 8-bit operation, the amount of storage required for character ranges in the compiled expression may be double that of the standard requirement.

### 4.3.6 The setlocale(S) Routine

The **setlocale** routine is used to read or set the international environment according to a specified category and locale.

At program startup the equivalent of the following call is executed:

**setlocale(LC_ALL, "")**

This is so that the initial locale for the program is defined as in the user environment.

### 4.3.7 The strftime(S) Routine

This routine converts and formats date and time information used by applications.

### 4.3.8 String Collation

String collating routines have been introduced to accommodate 8-bit character sets.

### 4.3.9 irand48() and krand48()

The functions **irand48**() and **krand48**() are not supported in this release.

## 4.4 Longest Allowed Path Names

The longest path name in a C program is restricted to 1024 bytes. System calls that require path names as arguments now fail, setting **errno** to **ENAMETOOLONG**, if a longer path name is given.

Previously, the path name was not restricted by the operating system; however, most programs gave an ad hoc limit to the length. Generally, these limits were well below 1024 bytes, so most programs should not be affected by this change.

Note also that any path component longer than 14 characters also causes **ENAMETOOLONG** to be set.

The **limits.h** file defines a macro **_POSIX_PATH_MAX** to be the longest length of a path name. Use *pathconf()* to determine **_POSIX_PATH_MAX** portably over various FFS filesystem types. In previous releases, this file incorrectly set the macro to 256, but it will probably be changed in a future release to 1024. Local system administrators can safely change the value for **_POSIX_PATH_MAX** to 1024 without harm, as the current system internally uses the longer limit.

You are encouraged to include the *limits.h* file with a statement like:

```
#include <limits.h>
```

and to refer to the **_POSIX_PATH_MAX** macro for the longest path name allowed.

## 4.5 malloc Issues

There are two versions of **malloc** distributed with the Development System. The standard **malloc** is contained in the file */lib/libc.a*. There is an alternate **malloc** in */lib/libmalloc.a*. Both are documented under the **malloc**(S) manual page.

If your program uses many **malloc** and **free** calls, running the program under the System V/386 Operating System can cause an excessive page swapping problem as **malloc** must search the entire list of allocated and free blocks. If you are using **malloc** and **free** calls extensively, it is suggested that you use the alternate **malloc** found in */lib/libmalloc.a*. To use */lib/libmalloc.a*, compile your program with the **-lmalloc** flag on your **cc**(CP) command line.

The alternate **malloc** maintains a separate list of free blocks and can be faster, but data in any allocated block is immediately overwritten when the block is declared free. The standard **malloc** preserves data between consecutive **free** and **malloc** calls. Some programs rely on this functionality of the standard **malloc**.

### 4.5.1 mallinfo()

Note that you must call **malloc**() at least once before calling **mallinfo**(). Otherwise, calling **mallinfo**() can result in unpredictable behavior.

## 4.6 Memory Models

The only memory model supported for 386 code is "small" model. Small model has two 32 bit segments; one for code and one for data. Small, middle, large, huge, and compact models are supported for 8086/286 code.

## 4.7 MINDOUBLE

The C compiler does not compute the value of the constant MINDOUBLE correctly. It always evaluates to 0 when used in expressions. This constant is defined in */usr/include/values.h*.

## 4.8 Preprocessor Names

Symbols which are used in **#define**, **#ifdef** and other preprocessor commands must now conform to the same rules as those for C identifiers. They must begin with an alphabetic character or an underscore and can only contain alphanumeric characters and the underscore character.

Previous versions of the C compiler allowed other non-alphanumeric characters such as "." in preprocessor symbols.

### 4.9 Stack Size

80386 programs have a variable sized stack that is expanded by the kernel as needed. 8086/286 programs run under the 386 Operating System have a fixed size stack. The default stack size for 8086/286 binaries is 4 Kilobytes unless the **-F** option of the linker was used. Once compiled, you can change the stack size of an 8086/286 program using the **-F** option to **fixhdr**(C).

### 4.10 Checking Storage Class in Declarations

The ANSI standard requires that declarations and definitions of functions and variables must have matching storage classes as well as matching types. Typically, this causes problems in code when a function is first used (and implicitly declares it as an "extern int") and is later defined as "static int." ANSI requires that the first use of the function be preceded by an explicit declaration. For example:

```
static int foo(); /*required by ANSI C */
```

*Program text*

```
        foo();

static foo()
```

The **-Me** option disables this strict checking of storage class.

### 4.11 String Constants

The 386 C compiler has been extended to allow 4Kbytes in string constants. The compiler generates the following message if this limit is exceeded:

```
string too big, trailing characters truncated.
```

## 4.12 Substituting Parameters in Quoted Strings

The proposed ANSI standard for C does not allow the substitution of macro formal parameters within quoted strings. However, many existing C compilers allow this. Consider the following macro definition and use:

```
#define      CTRL(c)              ('c' & 0x1f)
putchar(CTRL(g));
```

The above statement operates as intended when using the AT&T Compiler. The AT&T compiler expands this to:

```
putchar(('g' & 0x1f));
```

But according to the ANSI standard it must be:

```
putchar(('c' & 0x1f));
```

The Microsoft C compiler follows the ANSI standard strictly, therefore you must use this substitution as follows:

```
#define      CTRL(c)              (c & 0x1f)
putchar(CTRL('g'));
```

Then the compiler expands this to:

```
putchar(('g' & 0x1f));
```

The **-xpg2** option to the 386 C compiler allows the non-ANSI functionality described above, but displays a warning that a non-standard extension has been used.

## 4.13 terminfo curses

When you use terminfo curses in a 286 binary, you should compile with the **-F** option set to 2000 or greater to increase the fixed stack size. For small model programs, compile with the **-i** flag.

## 4.14 Third-Party Language Library

If you are using a third-party language product, such as Microsoft Pascal, you might experience errors when using the link editor supplied with the language product. This is due to the addition of a new library that is necessary for linking modules with C libraries.

This library is called **/lib/libh.a** and needs to be added explicitly to your link command in order to correctly compile your programs.

### 4.15 Tokens Following #else and #endif

The C compiler issues a warning if it finds anything other than blank space or comments following an **#else** or **#endif** preprocessor directive. Thus the first example below would produce a warning but the second would not.

```
#endif      M_I386
#endif      /* M_I386 */
```

These warnings are produced at the default level of 1 on both the 286 and 386 compilers.

### 4.16 Transforming Parameters to Strings

The ANSI standard defines a mechanism for transforming macro formal parameters into quoted strings (the stringizing operator, "#") and an operator for pasting strings, "##". These have been added to both the 286 and 386 preprocessors. A slight restriction on the 386 is that white space is not allowed between the stringizing operator and the formal parameter it operates on.

### 4.17 types.h

Some of the C language *.h* files require that *<sys/types.h>* be included first. An error message generally occurs when a type is used in an #include file but not declared. Often, the problem is corrected by including *<sys/types.h>* earlier in the program.

### 4.18 Variable Assignments

Assignments of the following form generate incorrect code when compiled with optimization enabled:

```
int i;
register  char *buf;

buf[ i + i ] = buf[ i ];
buf[ i ] = buf[ i + i ];
```

Note that if "buf" is not a register variable, or if the index

expression "i + i" contains any term other than "i", such as a constant or another variable, correct code is generated. Thus, both of these work correctly:

```
buf[ i + i + 1 ] = buf[ i ];
buf[ i + n ]     = buf[ i ];
```

### 4.19 Variable Declarations

The error `Segmentation Violation: core dumped` can occur if you declare too many variables within a single declaration statement. For example, the following text can cause the compiler to dump core:

```
char *
      x1,    /* comment */
      x2,    /* more comment */
      .
      .
      .
      x934; /* more comment */
```

Twenty-five variables in a declaration is a safe maximum. Break longer declarations into two or more statements to avoid this possible problem.

### 4.20 XENIX Cross Development Notes

The notes in this section refer to points of interest to C programmers developing for the XENIX environment only.

#### 4.20.1 386 Floating Point Emulation

286 floating point emulation software in 286 Operating Systems prior to 2.3 does not correctly execute floating point comparisons between the top of the floating point stack and memory. When you develop applications using the **-dos**, or the **-M0**, **-M1**, or **-M2** flags, you should use the **-Go** option to **cc**(CP) to instruct the compiler to allow for this. Note that the **-compat** flag implies the **-Go** option automatically.

When using masm, special care needs to be taken not to make floating comparisons with memory operands. To execute correctly on

previous 286 floating point emulators, only comparisons of different stack elements should be made. Note that pushing a memory operand onto the stack causes the source/destination relation between the two operands to be reversed in a subsequent comparison operation. As an example:

**fcom** *memory_address*
**jg** *label*

should be written as:

**fld** *memory_address*
**fcomp** *ST(1)*
**jl** *label*

### 4.20.2 XENIX Device Drivers

To develop installable device drivers suitable for use under XENIX, you must have the XENIX Development packages installed.

### 4.20.3 setvbuf(S)

The order of the parameters to **setvbuf**(S) has been changed to conform with the SVID. In releases of the XENIX Operating System prior to 2.3, the order of the parameters was:

**int setvbuf (stream, type, buf, size)**

In release 2.3 and later of XENIX and all releases of SCO System V, the order of the parameters is:

**int setvbuf (stream, buf, type, size)**

The files */lib/compat*[LMCS]*setvbuf.o* can be used for backwards compatibility.

# 5. SCO UNIX Development System Documentation Notes

## 5.1 STREAMS Manual Pages

exit (S)
fcntl (S)
getmsg (S)
ioctl (S)
open (S)
poll (S)
putmsg (S)
read (S)
signal (S)
sigset (S)
write (S)

These commands have features or references related to STREAMS. Additionally, The (NSL) and (STR) sets of manual pages are included in the streams documentation.

## 5.2 termcap(S) manual page

The **termcap**(S) manual page refers to the **stty**(S) manual page. This reference should be to **stty**(C). There is no **stty**(S) manual page.

## 5.3 atof(S) manual page

The following line at the top of the **atof**(S) manual page is incorrect:

```
#include <math.h>
```

The line should read:

```
#include <stdlib.h>
```

### 5.4 atexit(S) Manual Page

The **atexit**(S) manual page lists the incorrect return value. Upon successful completion, **atexit** returns a value of 0. On failure it returns a value of -1.

### 5.5 fieldtype(S) Manual Page

The final paragraphs under "Syntax" and "Functions" each have lines missing a letter. Where the text reads:

```
link_fieldtyp (type1,type2)
```

it should be:

```
link_fieldtype (type1,type2)
```

Note the added "e" at the end of "fieldtyp."

### 5.6 stopio(S) Manual Page

The **stopio**(S) manual page "Description" section states that **stopio** sends a SIGSYS signal. It should instead say that **stopio** sends a SIGHUP signal.

### 5.7 ioctl(S) Manual Page

The **ioctl**(S) manual page incorrectly states that "STREAMS errors are described in **streamio**(7)." Instead of **streamio**(7), it should reference **streamio**(STR), which is in the *Streams Programmer's Guide*, a part of the *Development System Developer's Guide*.

### 5.8 /usr/sys/conf Should Be /etc/conf/cf.d

In Appendix E of the *Streams Programmer's Guide* (in the *Development System Developer's Guide*), references to the file */usr/sys/conf/master* are incorrect. The correct filename is */etc/conf/cf.d/mdevice*.

### 5.9 memmove(S) Manual Page

The syntax for **memmove**(S) should direct you to include

```
#include <memory.h>
```

as well as

```
#include <string.h>
```

### 5.10 directory(S) Manual Page

To use *dirent* as described under "Syntax," you must also include *sys/ndir.h*.

### 5.11 SecureWare routines

In any source file using such SecureWare routines as **setluid**(S) and **getluid**(S), you must add the line:

```
#define SecureWare
```

Also, you must link such a file with libprot (using the **-lprot** option).

### 5.12 System Calls Requiring set_auth_parameters()

When you execute a program that uses or manipulates the protected password database, you might get the following error message:

```
Authentication database use not initialized first
```

This indicates that you need to make a call to **set_auth_parameters**() before attempting to access the protected password database. **set_auth_parameters**() is documented under **identity**(S).

### 5.13 cc(CP) Manual Page

The **-Ma** option to **cc**(CP) does not exist. If it is used, **cc** reports a command line error.

### 5.14 lint(CP) Manual Page

The **lint**(CP) manual page incorrectly lists several options as ".rm]B". The missing options are **-lx, -n, -p, -c**, and **-o**, in that order.

### 5.15 The sptalloc(K) Routine

The *Development System Device Driver Writer's Guide* states that the *base* argument to **sptalloc**(K) is a physical address. Instead, that argument is a page frame number.

### 5.16 STREAMS Overview

In the diagram on the first page of Chapter 9, "STREAMS," in the *Development System Device Driver Writer's Guide*, the upstream and downstream labelling and the read and write queue indicators are reversed, relative to the arrows.

### 5.17 t_accept(NSL) Manual Page

The syntax for **t_accept**(NSL) is missing a line. After the include statement should be

```
int t_accept(fd, resfd, call)
```

## 6. POSIX Implementation Defined Issues

Null pathnames are considered invalid and generate an error. (Rev. POSIX FIPS) p. 36

**chown**() is restricted to a process with appropriate privileges. C2 tuning may restrict all cases of chown to root. (Rev. FIPS)

Only a user with appropriate privileges can set file timestamps to arbitrary values using **utime**(). (Rev. FIPS)

System V/386 supports a value of NGROUPS_MAX greater than or equal to 8. (Rev FIPS)

**chown**() restricts changing the group-id of a file to egid or one of the supplementary group ids. (Rev. FIPS)

**exec**() saves the effective user and group id. (Rev FIPS)

**kill**() uses the saved set userid of the receiving process instead of the effective uid for determining eligibility of sending a signal. (Rev FIPS)

Upon **exit**() of a session process group leader, a SIGHUP is sent to each member of the session process group. (Rev FIPS)

Terminal special characters can be individually disabled using the value specified by POSIX_VDISABLE. (Rev. FIPS)

System V/386 supports POSIX_JOB_CONTROL. (Rev FIPS)

Support is provided for CPIO and USTAR data formats. (Rev FIPS)

Pathname components longer than NAME_MAX are invalid and generate an error. (Rev. FIPS)

EBUSY is supported as an errno value for **rename**(), **rmdir**(), and **unlink**(). (Rev. FIPS)

ISUID and ISGID are cleared by **chown**(), even for root process. Page 103.

Only a user with appropriate privileges can link or unlink directories. (Rev. FIPS)

The group ID of newly created files and directories is set to the group owner of the directory. (Revised POSIX FIPS).

System V/386 supports modem control on asynchronous serial terminal ports. (see the <config> parameters TERMIOS_TTY, TERMIOS_LOOP, and TERMIOS_TTY_NC. ) (Rev. POSIX FIPS)

System V/386 supports the environment variable HOME for the login shell as defined in Section 2.7 of the IEEE Std. 1988-1003.1. (Revised POSIX FIPS).

System V/386 supports the environment variable LOGNAME for the login shell as defined in Section 2.7 of the IEEE Std. 1988-1003.1. (Revised POSIX FIPS).

System V/386 supports **tcgetpgrp**() and _POSIX_JOB_CONTROL is defined. (Revised POSIX FIPS). See the "Get Foreground Process Group ID" in the Device- and Class-Specific Functions chapter of the POSIX standard.

System V/386 supports links to directories for user with appropriate privileges. (Revised POSIX FIPS). See the "Link to a File" in the Files and Directories chapter of the POSIX standard.

Access permission is required to access an existing file. (Revised POSIX FIPS).

System V/386 does not support links between different file systems. See the "Link to a File" entry in the Files and Directories chapter of the POSIX standard.

System V/386 supports using **unlink**() on directories from a user with appropriate privileges. (Revised POSIX FIPS). See the "Removing Directory Entries" entry in the Files and Directories chapter of the POSIX standard.

System V/386 allows unlinking a directory when it is being used by the system or another process. See the "Remove Directory Entries" entry in the Files and Directories chapter of the POSIX standard.

System V/386 allows the removal of a directory that is being used by another process. See the "Remove a Directory" section in the Files and Directories Chapter of the POSIX standard.

System V/386 does not require write permission for existing directories when renaming them. See the "Rename a File" section in the Files and Directories chapter of the POSIX standard.

System V/386 supports a call to **rename**() when the directory named is in use by the system or another process. See the "Rename a File" section in the Files and Directories chapter of the POSIX standard.

System V/386 supports the EINVAL error code for a call to **access**(). See the "File Accessibility" section in the Files and Directories Chapter of the POSIX standard.

System V/386 supports the setting of S_ISUID and S_ISGID by **chmod**(). See the "Change File Modes" section in the Files and Directories Chapter of the POSIX standard.

System V/386 does not support the EINVAL error for a call to **chown**() when an invalid "owner" argument is specified. See the "Change Owner and Group of a File" section in the Files and Directories chapter of the POSIX standard.

A call to **read**() or **write**() that is interrupted by a signal after it has successfully read or written any data returns the number of bytes read or written. (Revised POSIX FIPS). See "Input and Output Primitives" in the POSIX standard.

System V/386 supports character size of 5 bits, 6 bits, 7 bits, and 8 bits. See the "Control Modes" section in the Device and Class Specific Functions chapter of the POSIX standard.

System V/386 does not support changing the START and STOP special characters. See the "Special Characters" section in the Device and Class Specific Functions chapter of the POSIX standard.

System V/386 does not support the inclusion of the groupid with supplementary group IDs. See the "Get Supplementary Group IDs" section in the POSIX standard.

## 7. SCO 3.2 Product Engineering Toolkit Notes

The SCO Product Engineering Toolkit is a collection of utilities that aid developers in making their applications products easy to install and remove using the SCO **custom**(ADM) utility.

### 7.1 Feature Description

The SCO Product Engineering Toolkit provides the following major utilities:

| | |
|---|---|
| **mkperms** | generates a permissions list |
| **mkcuts** | prepares and moves distribution files from hard disk onto removable media |

| | |
|---|---|
| **hocheck** | compares the permissions list with the distribution files |
| **fdfit** | fits distribution files onto volumes |
| **fixperm** | corrects file permissions within permissions list |
| **pkgsize** | calculates the disk size of each distribution package |
| **volno** | generates volume numbers for each distribution file |

## 8. ISAM Development System Notes

### 8.1 Invoking SCO ISAM

There are two steps you must complete to use SCO ISAM with a C program:

1. Enter the following line with the other "include" statements at the top of your program:

   **# include <isam.h>**

2. Link the C program with the SCO ISAM libraries using the **-lisam** compiler option, as in the following example:

   **cc** *filename* **-lisam**

### 8.2 Converting C-ISAM files to SCO ISAM

SCO ISAM is call-compatible with C-ISAM applications as long as they use only X-OPEN ISAM calls. Such applications can be converted by linking them with SCO ISAM. However, SCO ISAM data and index files are not compatible with C-ISAM data and index files. To use C-ISAM applications, you must convert your C-ISAM data files to SCO ISAM data files.

If your data is in an Informix version 3.3 database and you have SCO Integra, you can use the "translate" program to convert your C-ISAM data files to SCO ISAM. (See the section "Converting Informix Files to Integra" in the *SCO Integra User's Guide* for instructions on using "translate.") However, if any of the Informix data files have a primary key (that is, "key primary" is specified in

the schema file), the translated files will not work with your existing C-ISAM applications. This problem occurs because Integra data files always contain an extra four-byte field that is guaranteed to be unique, while Informix data files contain an extra four byte field only if a primary key is not specified for the file. To avoid changing your existing applications, you should not use "translate" on any Informix file that has a primary key.

If you cannot use "translate," follow the procedure outlined below to convert your C-ISAM data and index files to SCO ISAM:

1. Write a C-ISAM program to unload your data into an ASCII delimited file. You can use **ldlong**(), **lddbl**(), **ldint**(), and **ldfloat**() to convert the values in the data files from C-ISAM format to numbers.

2. Rename the C-ISAM *.dat* and *.idx* files.

3. Write an SCO ISAM program to rebuild the files and indexes. If you have the code originally used to build the data and index files in C-ISAM, you can use the same code linked with SCO ISAM instead of C-ISAM.

4. Write an SCO ISAM program that loads the data stored in the ASCII delimited files into the SCO ISAM files you created in the previous step. You can use **stlong**(), **stdbl**(), **stint**(), and **stfloat**() to convert the numbers you read from the ASCII delimited files to SCO ISAM format.

If your C-ISAM applications do not link after you complete this procedure, it means your programs contain calls not defined in the X-OPEN ISAM specification. You need to rewrite these programs so that they use only X-OPEN calls.

## 8.3 Lock Manager Setup and Cleanup

To avoid the standard configuration limitations of kernel file locking (often as few as 50 locks in the kernel), a shared memory segment is used to contain lock tables. Access to this shared memory segment is regulated by a kernel semaphore. It is not necessary to do any explicit programming to use this scheme, but it does affect the cleanup procedure in abnormal situations.

### 8.3.1 Setup

You might want a separate lock manager table for each application accessing its particular sets of files. IPC resources (shared memory and semaphores, for example) in the UNIX system are identified by a "key." The key that SCO ISAM uses to set up the shared memory and semaphores is taken from the environment variable DBKEY. All processes you want to operate concurrently on the same set of files and records must have the same value for DBKEY.

To set the value of DBKEY:

- If you use the Bourne shell, include two lines in each user's *.profile* file:

  **DBKEY**=*value*
  **export DBKEY**

  There are no spaces around the equals sign.

- If you use the C shell, include this line in each user's *.login* file:

  **setenv DBKEY** *value*

In the above examples, *value* is the value to be assigned to DBKEY.

If two users accessing the same SCO ISAM files do not have the same DBKEY, there is no locking protection.

### 8.3.2 Cleanup

The first user to access a lock manager shared-memory segment automatically creates it; the last person to finish with it removes it. However, the operating system allows only the user who owns a shared-memory segment to remove it from the system. Therefore, if a different person does the cleanup, that person cannot remove the shared memory. Instead, the setuid-root binary "inipcrm" is executed to do the cleanup. This procedure is handled by the library code without any user intervention.

However, if an application exits because of a signal or a memory violation, or if the application program does not close all the SCO ISAM files, then this cleanup does not occur and SCO ISAM programs will not work correctly. The status of the IPC mechanisms can be checked by the "ipcs" program, and shared memory or semaphores can be removed by the "ipcrm" program. Refer to your SCO UNIX System V *User's Reference* for information on using either of these programs.

## 8.4 Open File Limits

In the current library, a maximum of 10 separate SCO ISAM files can be open at one time. However, it is possible to open a single physical file more than once and, therefore, to access the same file with different keys, saving logical position in the file for each key. The total maximum number of opens is 20.

The lock manager allows a total of 2000 record locks in the current configuration, as well as up to 100 file locks.

# Appendix A

# New and Revised Manual Pages

## A.1 New and Revised Manual Pages

The following new and revised manual pages are included in this Section:

Revised pages:

**intro**(K)
**tty**(K)
**vas**(K)
**codeview**(CP)

New pages:

**clrbuf**(K)
**nmi**(K)
**sigsetv**(S)
**catgets**(S)
**catopen**(S)
**dumpmsg**(CP)
**gencat**(CP)
**iconv**(CP)
**mar**(CP)

INTRO (K)

# Intro

lists manual page references

## Description

This section describes the manual page on which each kernel routine is found.

The following table summarizes the kernel routines. The columns indicate the routine name, the manual page on which the routine appears, a description, and a code that indicates the following values:

| Letter | Meaning |
|---|---|
| B | Use this routine only in a block driver |
| C | Use this routine only in a character driver |
| G | General; can be used in a block or character driver |
| I | Can be called from an initialization routine |
| X | Can be called from an interrupt routine |
| M | Macro |
| A | Written in Assembly language |

The section (K) routines are summarized in the following table:

| Kernel Routine | Manual Page | Description | Code |
|---|---|---|---|
| **all_io** | **all_io** | Determines if all processors can do I/O | GI |
| **bcopy** | **bcopy** | Copies bytes in kernel space | GAIX |
| **bdistributed** | **bdistributed** | Allows multiprocessing | BI |
| **brelse** | **brelse** | Releases a block buffer | B |
| **btoc** | **btoc** | Returns number of pages (clicks) | GMIX |
| **btoms** | **btoc** | Returns number of sectors | GMIX |
| **bzero** | **bzero** | Sets memory locations to 0 (zero) | GAIX |
| **canon** | **canon** | Processes raw input data from a tty device | CA |
| **can_doio** | **can_doio** | Sees if current CPU can do device I/O | G |
| **clrbuf** | **clrbuf** | Zeros a block I/O buffer | BIX |
| **cmn_err** | **cmn_err** | Displays message or panics the system | GIX |
| **copyin** | **copyin** | Copies bytes from user to kernel space | GA |
| **copyio** | **copyio** | Copies bytes to/from physical address | G |
| **copyout** | **copyin** | Copies bytes from kernel to user space | GA |

INTRO (K)

| Kernel Routine | Manual Page | Description | Code |
|---|---|---|---|
| **cpass** | **cpass** | Passes character from user space | G |
| **ctob** | **btoc** | Returns number of bytes | GMIX |
| **db_alloc** | **db_alloc** | Allocates physically contiguous memory | GI |
| **db_free** | **db_alloc** | Frees physically contiguous memory | GI |
| **db_read** | **db_read** | Transfers data from kernel virtual to physical address | G |
| **db_write** | **db_write** | Transfers data from physical to kernel virtual address | G |
| **delay** | **delay** | Delays process execution for specified time | G |
| **deverr** | **deverr** | Prints message on console | B |
| **disksort** | **disksort** | Adds block I/O request to device's queue | BX |
| **DISPLAYED** | **video** | Returns TRUE if screen displayed | CIX |
| **dma_alloc** | **dma_alloc** | Allocates a DMA channel | GIX |
| **dma_breakup** | **dma_breakup** | Sizes request into 512-byte blocks | B |
| **dma_enable** | **dma_enable** | Begins DMA transfer | GIX |
| **dma_param** | **dma_param** | Sets up a DMA controller chip for DMA transfer | GIX |
| **dma_relse** | **dma_relse** | Releases previously allocated DMA channel | GIX |
| **dma_resid** | **dma_resid** | Returns bytes not transferred for DMA request | G |
| **dma_start** | **dma_start** | Begins DMA transfer | G |
| **eisa_check** | **eisa_check** | Checks for EISA bus | G |
| **eisa_nvm** | **eisa_nvm** | Gets EISA ROM data | G |
| **emdupmap** | **emdupmap** | Duplicates channel mapping | G |
| **emunmap** | **emunmap** | Disables mapping on a channel | G |
| **flushtlb** | **flushtlb** | Flushes the translate lookaside buffer | GAIX |
| **fubyte** | **fubyte** | Gets a character from user data space | GA |
| **fuword** | **fuword** | Gets a 32-bit word from user data space | GA |

INTRO (K)

| Kernel Routine | Manual Page | Description | Code |
|---|---|---|---|
| **getablk** | **geteblk** | Gets empty buffer from free list | B |
| **getc** | **getc** | Gets a character from a clist | C |
| **getcb** | **getc** | Gets cblock from clist | C |
| **getcbp** | **getc** | Gets characters from a clist | C |
| **getcf** | **getc** | Gets a cblock from free list | C |
| **getchar** | **getchar** | Gets one character of input | G |
| **geteblk** | **geteblk** | Gets empty buffer from free list | B |
| **inb** | **inb** | Reads a byte from an I/O address | GAIX |
| **ind** | **ind** | Reads double words from an I/O address | GAIX |
| **inw** | **inw** | Reads a 16-bit word from a physical I/O address | GAIX |
| **intralloc** | **intralloc** | Returns MPX handle | G |
| **iodone** | **iodone** | Signals I/O completion | BIX |
| **iowait** | **iowait** | Wait for I/O completion | B |
| **ktop** | **ptok** | Returns physical address from kernel | GMIX |
| **lockb** | **lockb** | Locks critical code section | G |
| **longjmp** | **longjmp** | Restores previously saved process context | GA |
| **major** | **major** | Returns major number from the device number | GM |
| **makedev** | **major** | Returns device number from major and minor numbers | GM |
| **memget** | **memget** | Allocates contiguous memory at initialization | GI |
| **minor** | **major** | Returns minor device number from the device number | GM |
| **nmi** | **nmi** | Detects a non-maskable interrupt | GIX |
| **outb** | **inb** | Writes a byte to an I/O address | GAIX |
| **outd** | **ind** | Writes double words to a physical I/O address | GAIX |
| **outw** | **inw** | Writes a 16-bit word to a physical I/O address | GAIX |

INTRO (K)

| Kernel Routine | Manual Page | Description | Code |
|---|---|---|---|
| **paddr** | **paddr** | Returns virtual address pointer to block data | BIX |
| **panic** | **panic** | Halts the system | GIX |
| **passc** | **cpass** | Passes character to user space | G |
| **physck** | **physio** | Verifies I/O request size | B |
| **physio** | **physio** | Performs physical I/O | B |
| **pio_breakup** | **pio_breakup** | Breaks up programmed I/O requests | B |
| **printcfg** | **printcfg** | Displays driver initialization message | GI |
| **printf** | **printf** | Print a message on the console | GIX |
| **psignal** | **psignal** | Sends signal to a process | CIX |
| **ptok** | **ptok** | Returns kernel address from physical | GMIX |
| **putc** | **putc** | Puts character on clist | C |
| **putcbp** | **putc** | Puts characters on clist | C |
| **putcb** | **putc** | Puts cblock on clist | C |
| **putcf** | **putc** | Puts cblock on free list | C |
| **putchar** | **putchar** | Prints a character on the console | G |
| **repinsb** | **repins** | Moves bytes to memory from I/O address | GA |
| **repinsd** | **repins** | Moves words to memory from I/O address | GA |
| **repinsw** | **repins** | Moves double words to memory from I/O address | GA |
| **repoutsb** | **repins** | Moves bytes from memory to I/O address | GA |
| **repoutsd** | **repins** | Moves words from memory to I/O address | GA |
| **repoutsw** | **repins** | Moves double words from memory to I/O address | GA |
| **scsi_get_gen_cmd** | **scsi** | Fills a command block | GIX |
| **scsi_getdev** | **scsi** | Gets SCSI device number | GIX |
| **scsi_mkadr3** | **scsi** | Makes 3-byte address | GIX |
| **scsi_s2tos** | **scsi** | Converts 2 bytes to short | GIX |
| **scsi_s3tol** | **scsi** | Converts 3 bytes to long | GIX |
| **scsi_stok** | **scsi** | Converts 3 bytes to address | GIX |
| **scsi_swap4** | **scsi** | Swaps 4 bytes | GIX |

INTRO (K)

| Kernel Routine | Manual Page | Description | Code |
|---|---|---|---|
| **selfailure** | **select** | Fails condition | G |
| **selsuccess** | **select** | Okays condition | G |
| **selwakeup** | **select** | Okays failed condition | G |
| **seterror** | **seterror** | Sets u.u_error with error code | G |
| **signal** | **signal** | Sends a signal to a process | CIX |
| **scsi_stol** | **scsi** | Converts 4 bytes to long | GIX |
| **sleep** | **sleep** | Suspends processing temporarily | G |
| **spl0** | **spl** | Permits all interrupts | GAIX |
| **spl1** | **spl** | Blocks context switch interrupts | GAIX |
| **spl2** | **spl** | Blocks level 2 interrupts | GAIX |
| **spl3** | **spl** | Blocks level 3 interrupts | GAIX |
| **spl4** | **spl** | Blocks level 4 interrupts | GAIX |
| **spl5** | **spl** | Blocks block device interrupts | GAIX |
| **spl6** | **spl** | Blocks character device and the clock's interrupts | GAIX |
| **spl7** | **spl** | Blocks all interrupts | GAIX |
| **splbuf** | **spl** | Blocks buf access interrupts | GAIX |
| **splcli** | **spl** | Blocks clist access interrupts | GAIX |
| **splhi** | **spl** | Blocks all interrupts | GAIX |
| **splni** | **spl** | Blocks network interrupts | GAIX |
| **splpp** | **spl** | Blocks ports board interrupts | GAIX |
| **spltty** | **spl** | Blocks tty interrupts | GAIX |
| **splx** | **spl** | Enables previous spl level | GAIX |
| **sptalloc** | **sptalloc** | Allocates temporary memory or maps a device into memory | GI |
| **sptfree** | **sptfree** | Releases memory previously allocated with sptalloc | GI |
| **startio** | **startio** | Runs **xxstart** routine | G |
| **strcat** | **string** | Appends strings | GIX |
| **strncat** | **string** | Appends strings of *n* characters | GIX |
| **strcmp** | **string** | Compares strings | GIX |
| **strncmp** | **string** | Compares strings of *n* characters | GIX |
| **strcpy** | **string** | Copies strings | GIX |
| **strncpy** | **string** | Copies strings of *n* characters | GIX |
| **strlen** | **string** | Returns string length | GIX |
| **strchr** | **string** | Returns character position | GIX |
| **subyte** | **subyte** | Stores a character in user data space | GA |
| **suser** | **suser** | Determines if current user is the super-user | G |
| **suword** | **suword** | Stores a 32-bit word in user data space | GA |

INTRO (K)

| Kernel Routine | Manual Page | Description | Code |
|---|---|---|---|
| **timeout** | **timeout** | Schedules a time to execute a routine | GX |
| **ttclose** | **tty** | Closes access to tty device | C |
| **ttin** | **tty** | Gets data from receive buffer | CX |
| **ttinit** | **tty** | Initializes line discipline | C |
| **ttiocom** | **ttiocom** | Interpret tty driver I/O control commands | C |
| **ttiwake** | **tty** | Awakens input requests | CX |
| **ttopen** | **tty** | Opens tty device | C |
| **ttout** | **tty** | Puts data into transmit buffer | CX |
| **ttowake** | **tty** | Awakens output requests | CX |
| **ttrdchk** | **tty** | Verifies characters to read | C |
| **ttread** | **tty** | Copies tty data to user space | C |
| **ttrstrt** | **tty** | Restarts tty access | C |
| **ttselect** | **tty** | Ensures read or write without block | C |
| **tttimeo** | **tty** | Times input request | C |
| **ttwrite** | **tty** | Copies data from user | C |
| **ttxput** | **tty** | Puts data into output queue | C |
| **ttyflush** | **tty** | Releases queue contents | C |
| **ttywait** | **tty** | Waits for UART to drain | C |
| **unlockb** | **unlockb** | Unlocks critical code section | G |
| **untimeout** | **timeout** | Cancel scheduled timeout request | GX |
| **vasbind** | **vas** | Binds virtual address to physical | GI |
| **vasmalloc** | **vas** | Allocates virtual user memory | GI |
| **vasmapped** | **vas** | Releases allocated memory | GI |
| **vasunbind** | **vas** | Unbinds bound memory | GI |
| **viddoio** | **video** | Supports I/O controls | C |
| **vidinitscreen** | **video** | Initializes multiscreen | CI |
| **vidmap** | **video** | Maps memory | CIX |
| **vidresscreen** | **video** | Restores screen | CX |
| **vidsavscreen** | **video** | Saves screen | CX |
| **vidumapinit** | **video** | Maps user memory | CI |
| **vidunmap** | **video** | Unmaps memory | CIX |
| **vtop** | **vtop** | Convert a virtual address to a physical address | GX |
| **wakeup** | **wakeup** | Wakes up a sleeping process | GX |

TTY (K)

# tty: ttclose, ttin, ttinit, ttiwake, ttopen, ttout, ttowake, ttrdchk, ttread, ttrstrt, ttselect, tttimeo, ttwrite, ttxput, ttyflush, ttywait

tty driver routines

## Syntax

```
#include "sys/types.h"
#include "sys/tty.h"

int
ttclose(tp)
struct tty *tp;

int
ttin(tp, code)
struct tty *tp;
int code;

void
ttinit(tp)
struct tty *tp;

int
ttioctl(tp, cmd, arg, mode)
struct tty *tp;
int cmd, arg, mode;

void
ttiwake(tp)
struct tty *tp;

int
ttopen(tp)
struct tty *tp;


int
ttout(tp)
struct tty *tp;
```

TTY (K)

```
void
ttowake(tp)
struct tty *tp;

int
ttrdchk(tp)
struct tty *tp;

int
ttread(tp)
struct tty *tp;

void
ttrstrt(tp)
struct tty *tp;

void
ttselect(tp, rw)
struct tty *tp;
int rw;

void
tttimeo(tp)
struct tty *tp;

int
ttwrite(tp)
struct tty *tp;

void
ttxput(tp, ucp, ncode)
struct tty *tp;
int ncode;
union {
        unsigned short ch;
        struct cblock *ptr;
} ucp;

void
ttyflush(tp, rdwrt)
struct tty *tp;
int rdwrt;

void
ttywait(tp)
struct tty *tp;
```

TTY (K)

## Description

The routines are:

*ttclose* — Called by the line discipline zero *l_close* routine to remove access to a tty device from the process that called it. *ttclose* disables ISOPEN from the **t_state** member of the **tty** structure, calls *ttioctl* with the LDCLOSE argument, and disables XCLUDE from the **t_lflag** member of the **tty** structure.

*ttin* — Called by the line discipline zero *l_input* routine to get characters from a TTY device. *ttin* is called in a driver's interrupt routine to process and move characters from **t_rbuf** to the raw character queue, **t_rawq**. *ttin* processes the *termio*(M) **c_cc** values of VINTR, VQUIT, VSUSP, VSWTCH, VEOL, VERASE, VKILL, and VEOF. In addition, *ttin* checks that ICANON is set when processing the **c_cc** VMIN and VTIME values. *ttin* performs input escape mapping for internationalization character processing. If the *code* argument to *ttin* is L_BREAK, *ttin* sends the SIGINT signal to all associated processes. *ttin* then calls *ttyflush* to release both read and write buffers, and returns if no characters are found in **t_rbuf**. If either ECHO is set or after processing international character mapping, *ttin* calls the driver's *xxproc* routine with T_OUTPUT set. *xxproc* can also be called with T_SWTCH set if VSWTCH is enabled and if NOFLSH is disabled. *ttin* calls these other tty routines: *ttyflush, ttxput, tttimeo*, and *ttiwake*.

*ttinit* — Initializes the **tty** structure for line discipline 0 (zero). The following members of the **tty** structure are initialized:

- **t_line** — line discipline index is set to 0 (zero)
- **t_iflag** — terminal input control is set to 0 (zero)
- **t_oflag** — terminal output control is set to 0 (zero)
- **t_lflag** — line discipline terminal control is set to 0 (zero)

TTY (K)

- **t_cflag** — terminal hardware control is set to these values: 1200 baud (SSPEED variable), 8 bits (CS8 variable), enable receiver (CREAD variable), hang up on last close (HUPCL variable)

*ttioctl* Used to allocate, deallocate, or move the contents of terminal buffers. Called through the *l_ioctl* function of the line switch table. The *cmd* argument to *ttioctl* has the following values:

- LDOPEN — allocate a receive buffer, initialize several **t_rbuf** members (**c_ptr, c_count,** and **c_size**), and call the driver's *xxproc* routine with T_INPUT as the argument.

- LDCLOSE — call the driver's *xxproc* routine with the T_RESUME argument; call *ttywait* to wait for data to clear the UART; and call *ttyflush* to empty the **t_canq** and **t_rawq** buffers, as well as the contents of **t_rbuf**.

- LDCHG — move contents of the raw queue, **t_rawq**, to the canonical queue, **t_canq**. LDCHG only works if ICANON is enabled.

*ttiwake* Called from an interrupt routine to wake up any processes that are asleep waiting for characters to appear in the raw input queue, **t_rawq**. *ttiwake* only works when IASLP is set in the **t_state** field of the **tty** structure. *ttiwake* disables IASLP.

*ttopen* Called on each device open to initiate access to a tty device. Determines the process group leader and then opens a tty device for I/O. This routine requires user context and cannot be called from an interrupt routine. The verification of the process group leader includes a check to determine if the device was opened with the POSIX FNOCTTY extension. If FNOCTTY is not specified when the device is opened, then the process wants to have a controlling tty assigned by *ttopen*. Conversely, if FNOCTTY is specified on the open, then the process is flagged to not have a controlling tty. FNOCTTY is defined in **sys/file.h** and is sent by the kernel to an *xxopen* routine as an open file flag.

If the process is the process group leader and a controlling tty is requested, then **u.u_ttyp** is set to the address of the **t_pgrp** field in the device's *tty* structure, and **t_pgrp** is set to the processes' **p_pgrp** value in the *proc* structure. The *user* structure is

TTY (K)

defined in **sys/user.h** and the *proc* structure is defined in **sys/proc.h**.

If exclusive super user control is requested with XCLUDE set in **t_lflag**, then all other open requests are returned with EBUSY set in **u.u_error**. When access is cleared for an open, *ttioctl* is called with LDOPEN to open the device. After this step concludes, **t_state** is cleared of a previously set WOPEN value and ORed with ISOPEN.

*ttout*

Gets next packet of characters from the **t_outq** output queue and moves them to the **t_tbuf** transmit buffer for output to the terminal device. This routine is called from an interrupt routine. *ttout* performs timing to wait for packets to be received if a delay value is set in **t_schar**. This value is either FALSE if no timing is required, or set to a specified number of clock ticks. The range of ticks that can be specified is decimal 6 to 133. *ttout* ensures this range by ANDing the specified value with octal 0177 and then adding 6. The actual timed interval varies by the number of clock ticks that occur in a second. This value is contained in the manifest constant HZ.

If characters are not found in the **t_outq** output queue and if **t_state** contains TTIOW, then wake up any jobs waiting for output to the terminal to drain. Before waking the sleeping processes (that are sleeping on the address of **tp->t_oflag**), *ttout* disables TTIOW.

If further delay processing is requested by **t_state** containing EXTPROC, or **t_xstate** containing EXTDLY, or **t_oflag** not containing OPOST, then packet movement is performed character by character. Then the **c_count, c_size,** and **c_ptr** fields of the *cblock* structure contained by **t_tbuf** are set respective to the number of bytes moved. When character timing is in effect, the character taken from the output queue is checked to see if it is greater than the value of the QESC constant. If the character is greater than QESC, then the character is a **timeout** tick.

If the count of the characters in the output queue falls below the low water mark (TTLOWAT), then *ttowake* is called.

*ttout* returns the value of CPRES (typically octal 100000—defined in **sys/tty.h**) when EXTPROC is set to indicate further processing needs to be handled by

TTY (K)

the controller, or when OPOST is set to indicate that further processing needs to be handled by the software. If neither of these flags are set, zero (0) is returned.

*ttowake* Called from an interrupt routine to wake up any processes that are asleep waiting for characters to appear in the output queue, **t_outq**. *ttowake* only works when OASLP is set in the **t_state** field of the **tty** structure. *ttowake* disables OASLP.

*ttread* Called from a driver's *xxread* routine indirectly through the *l_read* line discipline switch function. *ttread* conveys characters processed by *canon* from the canonical queue, **t_canq**, to the user process.

*ttrdchk* Returns a non-zero value if there are characters to be read. If carrier is present (CARR_ON is enabled in **t_state**), *ttrdchk* tests **t_canq** for characters. **t_canq** is the queue for characters that have been processed by the *canon* routine. If characters are present, a 1 is returned. If characters are not present, and *canon* is being used to process characters from the terminal (ICANON is set in **t_lflag**), then *ttrdchk* tests the delimiter count to see if a delimiter has been entered on the tty device. The delimiter count is held in **t_delct** in the **tty** structure. If a delimiter was received, 1 is returned. If characters are not present, but *canon* is not being used to process characters, then the raw character queue, **t_rawq**, is checked for characters. If characters are waiting in the raw queue, 1 is returned.

*ttrstrt* Restart tty device output after a delay timeout. *ttrstrt* performs only one task; it calls the driver's *xxproc* routine with arguments of *tp* and T_TIME. (*tp* is the pointer to the *tty* structure passed through from the argument to *ttrstrt*.)

*ttselect* Ensure that a read or write can be performed with no blocking. (If blocking occurs, the process will be put to sleep until the I/O can be satisfied.) Do not call *sleep*(K) before calling *ttselect*, and do not call *ttselect* from an interrupt routine. *ttselect* has two modes determined by its *rw* argument. These modes are SELREAD and SELWRITE.

*tttimeo* Satisfy VTIME timing requirement for data input. *tttimeo* does not execute if ICANON is set, if VTIME or VMIN are not set, or if there aren't any characters to process in the raw queue, **t_rawq**. *tttimeo* calls *timeout*(K) for the time specified in VTIME times the

TTY (K)

number of ticks per second (Hz) divided by 10. While *timeout* is active, **t_state** is ORed with RTO and TACT. When the timing finishes, *ttiwake* is called to awaken any processes that are sleeping on the raw input queue.

*ttwrite*

Called from the line discipline *l_write* function call in a driver's *xxwrite* routine to copy data from the user program into the driver so that the data can be displayed on a tty device. If there is no carrier, *ttwrite* returns immediately. *ttwrite* requires user context and therefore cannot be called from an interrupt routine. If a paging fault occurs while attempting to get the data from user space, **u.u_error** is set to EFAULT. In the course of the copy operation, **u.u_base** and **u.u_count** are updated to reflect the amount of data transferred. *ttwrite* calls *ttxput* to write the data to the terminal.

Before data is fetched from user space, the number of characters in the output queue, **t_outq**, are checked to see if the count exceeds the high water mark. The high water mark is the point at which tty input is suspended so that the characters coming in don't overload the tty driver's ability to process them and echo them to output. If the count exceeds the high water mark, the driver's *xxproc* routine is called with the T_OUTPUT argument, and the process is put to sleep until the tty driver can handle I/O again. The process sleeps on the address of **t_outq** at a priority of TTOPRI (above PZERO) and can be awakened prematurely by a signal. If a signal is sent to the sleeping process, a *longjmp*(K) occurs, returning control to the calling user process and putting EINTR in **u.u_error** (*errno* in user space). If a write occurs in the background, SIGTTOU is sent to all processes in the current process group.

*ttxput*

Puts characters on the tty output queue (**t_outq**), adds delays, expands tabs, and handles carriage return and new line characters. *ttxput* is called from both base level to output characters to a tty device and from interrupt level for echoing characters read in from the tty device. If the queue escape character, denoted as QESC, is detected, it is put directly on the output queue. The next character after QESC is treated as a timer character if the octal value is greater than 200. The timer character is used in a *timeout*(K) call in *ttout*. If the character after QESC is less than 200, it is treated as an ordinary character to be output.

TTY (K)

If **t_state** contains EXTPROC, but **t_lflag** does not contains XCASE, then no post processing is required. Additionally in this state, characters can be internationally mapped if requested, and shipped to the output queue for further processing. XCASE indicates that special characters should be "escaped" by being preceded with a backslash. Refer to *termio*(M) for a list of characters that are translated when XCASE is specified. International character mapping is requested if **t_mstate** is true. If XCASE processing is required, then international character mapping can also be requested, and processing for QESC is available (QESC handling is not available when no post processing is required).

*ttxput* processes characters with octal values greater than 200 as special characters and performs delay processing if **t_state** does not contain EXTPROC. If EXTPROC is set, than delay processing is assumed to be handled by an external process. Characters that may require translation and also need to indicate a delay are shown in the following table:

| Octal Value | Description |
|---|---|
| 0201 | non-printing character |
| 0202 | backspace |
| 0203 | line feed |
| 0204 | tab |
| 0205 | vertical tab |
| 0206 | carriage return |
| 0207 | form feed |

If delay handling is being provided by *ttxput*, then the delay is calculated based on the value of **t_oflag**. The *ttout* routine does the actual timed delay. *ttxput* outputs the QESC character as well as the timing value ORed with octal 0200.

*ttyflush* Called to release character blocks to the free list from the write buffer, **t_outq**, or from the two read buffers, **t_canq** and **t_rawq**. The *rdwrt* argument to *ttyflush* should be ANDed with either FREAD to release read buffers, or with FWRITE to release the write buffer. When the blocks in the write buffer are released, the driver's *xxproc* routine is called with the T_WFLUSH argument. If **t_state** contains OASLP, any processes sleeping on the address of **t_outq** are awakened and OASLP is disabled, and if **t_state** contains TTIOW, then TTIOW is disabled and all processes sleeping on the address of **t_oflag** are awakened.

TTY (K)

If the blocks in the read buffers are being released, then the driver's *xxproc* routine is called with the T_RFLUSH argument. If **t_state** contains IASLP, IASLP is disabled and all processes sleeping on **t_rawq** are awakened.

*ttywait* Called to drain the contents of the universal asynchronous receiver/transmitter (UART). *ttywait* waits 11 bit times for the UART to empty of all data. The baud rate is taken from **t_cflag&CBAUD** and possible values are (in bits per second): 0.5, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, or 38400. If upon entry to this routine, characters are in **t_outq**, or **t_state** contains BUSY or TIMEOUT, then **t_state** is ORed with TTIOW and the process is put to sleep on the address of **t_oflag** at TTOPRI. TTOPRI has a value above PZERO and therefore can be prematurely awakened by a signal. Should a signal occur, control returns to the user program and the EINTR error code is placed in **u.u_error** and *errno* in the user program.

## Parameters

*tp* A pointer to the **struct tty** data structure associated with the device being accessed.

*code* Used by *ttin* and can be set to L_BREAK to cause all processes associated with the terminal to be sent a SIGINT signal, and to have all character blocks on **t_outq, t_canq,** and **t_rawq** queues be released to the free list.

*cmd, arg,* or *mode* Used by *ttioctl* to determine which I/O control command is being requested.

*rw* Used by *ttselect* to determine whether the driver is testing for read or write access without blocking. Possible values are SELREAD or SELWRITE.

*ucp, ncode* Used by *ttxput*. *ucp* describes a union of a character or a pointer to a block of characters. *ncode* is a flag used to indicate which value *ucp* contains; if *ncode* is zero (0), *ucp* is a character; any other value indicates that *ucp* is a pointer to a **cblock**.

TTY (K)

*rdwrt* — Used by *ttyflush* to indicate which buffers to release to the free list. AND with FREAD to indicate that the **cblock**s in **t_canq** and **t_rawq** should be released; AND with FWRITE to indicate that the blocks in the **t_outq** should be released.

## Notes

All routines on this manual page can be used only with character device drivers.

## Return Values

Only *ttout*, *ttrdchk*, and *ttread* return meaningful values. These values are described as follows:

| Routine | Value | Event |
|---|---|---|
| ttout | 0 | Either a delay was satisfied or all characters have been moved from **t_outq** to **t_tbuf** |
| | 0100000 (octal) | When external processing is requested by EXTPROC to be handled by a hardware controller, or by OPOST to be handled by the software. |
| ttrdchk | 0 | All characters have been read |
| | 1 | 1. If characters remain to be read<br>2. If no characters are in **t_canq**<br>3. If canonical processing is requested and a line delimiter has been received. |

The following routines are cast as having an **int** return value, but the return values are indeterminate: *ttclose*, *ttin*, *ttioctl*, *ttopen*, *ttread*, and *ttwrite*.

TTY (K)

The **u.u_error** field is set under the following circumstances:

| Routine | Value | Event |
|---|---|---|
| ttioctl | EINVAL | An improper **cmd** argument to *ttioctl* was specified |
| ttopen | EBUSY | A process trying to open a tty device is not super user and XCLUDE is set in **tp->t_lflag**. |
| ttread | EFAULT | Copying data to user space failed during a *copyout*(K) or *subyte*(K) call |
| ttwrite | EFAULT | 1. Copying data from user space failed during a *copyin*(K) or *fubyte*(K) call<br>2. A negative number was detected in the data received from user space |

## See Also

termio(M), canon(K), ttiocom(K)

CODEVIEW (CP)

# codeview

visual debugger

## Syntax

**cv** [options] [executable_file [arguments]]

## Description

Codeview is a screen-oriented debugger. Codeview can operate in either "window mode" or "sequential (serial) mode" for a simple line interface. In window mode, pull-down menus and simple keystrokes offer fast access to most common commands.

In addition to numerous standard debuging functions, codeview permits browsing of source or assembly code, indicates which line will be executed next, highlights breakpoints, switches screens to display program output, dynamically watchs the value of variables (local or global), offers fast tracepoints using hardward debug registers, and provides a variety of ways for conveniently examining and modifying process text and data. Codeview can debug programs written in C, FORTRAN, BASIC, or Pascal as well as assembly language. Codeview will debug both COFF and x.out format executable files. Codeview provides support for interpreting core files and extends full control of process signals to the user.

To full take advantage of codeview's debugging features, the executable file should be compiled and linked with the correct options for generating symbolic debug and line number records. The executable should at least have a standard symbol table (the executable should not be stripped). Otherwise, codeview can only provide assembly level debugging, without symbol names. It is possible for only a portion (a single module) of the executable to be compiled with symbolic debug support. The correct options for compiling and linking for use with codeview are described in the volume titled *The Codeview Debugger* .

When run without arguments, codeview looks for files named **a.out** and **core** in the current directory, and uses them as the executable and core files. To cause codeview to ignore a core file in the current directory, use the option **-C--** .

Any command line parameters following the executable file name will be passed as command line arguments to the process. Once codeview is running, arguments can be reset using 'l' (restart).

CODEVIEW (CP)

Codeview takes the following options:

**-c** *command-list*
Execute *comand-list* on startup.

**-t** Run in serial mode.

**-C** *corefile* | **--**
Load core file *corefile*. Use **-C --** to ignore *core* in the current directory.

**-b** Specifies a monochrome (black & white) display.

This manual page is designed to be used as a reference by codeview users. For detailed information about how to use codeview, see the volume titled *The Codeview Debugger* .

CODEVIEW (CP)

## Command Syntax

| Pull Down Menu | |
|---|---|
| **file** | |
| open | open arbitrary text file for viewing |
| shell | shell escape |
| exit | exit codeview |
| **view** | |
| source, mixed, assembly | select source mode |
| registers | toggle register window |
| local/watch | toggle local/watch window |
| output | toggle to program output |
| **search** | |
| find | search for regular expression |
| next, previous | use last regular expression |
| label | goto text label |
| **run** | |
| start | re-load and go |
| restart | re-load |
| execute | slow-motion execute |
| clear breakpoints | remove all breakpoints |
| **watch** | |
| add watch | add watch expression |
| watchpoint | add watchpoint |
| tracepoint | add a tracepoint |
| delete watch | delete watch expression |
| delete all watch | delete all watch, watchpoints and tracepoints |
| **options** | |
| save output | toggle flip/swap option |
| bytes coded | toggle assembly mode verbosity |
| case sense | toggle symbol case sensitivity |
| **language** | |
| auto | |
| basic | |
| C | |
| fortran | |
| **calls** | |
| stack | |
| backtrace | |
| window | |
| **help** | |
| variety of help topics | |

In full screen mode, function keys perform the following operations:

CODEVIEW (CP)

| Function Keys | |
|---|---|
| F1 | Help |
| F2 | Toggle register window |
| F3 | Switch between source, assembly and mixed mode |
| F4 | Toggle between codeview screen and application output |
| F5 | Go |
| F6 | Toggle between source and dialog window |
| F7 | Go to cursored line |
| F8 | Trace (e.g. into functions) |
| F9 | Toggle breakpoint at cursored line |
| F10 | Step (e.g. over functions) |
| F12 | Menu selection |

Coveview understands the following special keystrokes:

| Special Keystrokes | |
|---|---|
| DEL | Interrupt current command |
| ESC | Cancel menu option; clear dialog command line |
| HOME | Top of screen; top of dialog command buffer |
| END | Bottom of screen; bottom of dialog command buffer |
| <alt>*letter* | Menu selection |
| <shift>F2 | Local/watch window toggle |
| <ctrl>F | Find regular expression |
| <ctrl>w | Add watchpoint |
| <ctrl>u | Delete watchpoint |

Dialog commands use the following syntax:

*command*[*arguments*][;*command*]

Dialog comands are not case sensitive. Symbol names may be case sensitive.

Many codeview commands take addresses or address ranges as arguments. For most commands, the following address constructs are valid:

| Address Constructs | |
|---|---|
| symbol | *symbolname* |
| register | [@]*register* |
| address | [*segment*:]*offset* |
| range | *startaddress endaddress* |
| | *startaddress* L *count* |
| | *startaddress* (uses a default count, usually 128) |
| line number | .[*filename*:]*linenumber* |

Several codeview commands take expressions as arguments. Expressions can be complex C expressions using constants, symbols, operators, casts and function calls. Expression evaluation corresponds to that of the C compiler including precedence and type conversions.

CODEVIEW (CP)

Constants use the following syntax:

| Constant Syntax | |
|---|---|
| *digits* | default radix |
| 0*digits* | octal |
| 0x*digits* | hexidecimal |
| 0n*digits* | decimal |
| "*chars*" | null terminated string |

Registers may also be referred to directly for reference, addressing and assignment using standard names (such as "ds," "ebp," and "ah"). If a symbol uses the same name as a register, the symbol name overrides the register name. In this case, the register can be specified using "@" (example: @ds).

The following assembly style directives can be used with register names for addressing instead of brackets:

| | |
|---|---|
| BY | byte pointer |
| WO | word pointer |
| DW | double word pointer |

For example, **WO si+bp+6** is equivalent to **WORD PTR [si][bp+6]**.

## Code Display

The following table lists the commands that are used to view code:

| Code Display | |
|---|---|
| set source mode | s [+\|-\|&]<br>+ C source<br>- assembly<br>& mixed<br>with no operand, displays current mode |
| view source | v [*address*] |
| search for *regular expression* | /*regular expression* |
| Move in the source window | <pgup>, <pgdn><br>arrow keys<br>HOME, END |
| unassemble code | U [*address* \| *range*] |

CODEVIEW (CP)

## Data Display

The following commands display data:

1. ?*expression*[,*format*]

   This table lists the valid format specifiers:

| Format Specifiers | |
|---|---|
| d,i | signed decimal integer |
| u | unsigned decimal integer |
| o | unsigned octal integer |
| x,X | unsigned hexadecimal integer |
| f | floating point |
| e,E | scientific notation floating point |
| g,G | scientific notation floating point |
| c | character |
| s | null terminated string |
| h | used with d,o,u,x and X. Example: **,hd** (short) |
| l | used with d,o,u,x and X. Example: **,ld** (long) |

2. ??*expression*

   The following special keystrokes are recognized:

   | | |
   |---|---|
   | <return> | de-reference current object |
   | <esc> | return to dialog line |

3. r [*register*]

   This causes a register dump; without a register name, it shows all registers and disassembles the current instruction at cs:ip.

4. 7

   Math co-processor register dump.

CODEVIEW (CP)

## Variable and Memory Assignment

Use the following commands to assign values to variables or registers:

| Variable & Memory Assignment | |
|---|---|
| ?*expression* = *expression* | Left *expression* must evaluate to an lvalue. |
| r *register* [*value*] | Assign value to indicated register. If *value* is omitted, prompt for a new value. |

## Special Commands

The D (Dump) command displays memory contents, and uses this syntax:

D[*type*] [*address*|*range*]

The following types are recognized:

| Types | |
|---|---|
| B | bytes |
| A | ASCII characters |
| I | two-byte integers |
| U | unsigned two-byte integers |
| W | two-byte words, hexadecimal |
| D | four-byte words, hexidecimal |
| S | short reals (four-bytes) |
| L | long reals (eight-bytes) |
| T | ten-byte reals |

The C (Compare) command compares two memory regions and lists the differences. Use this syntax:

C *range1 address2*

The S (Search) command searches a memory range for a pattern, and lists the occurances of the pattern. Use this syntax:

S *range list*

The E (Enter) command modifies memory contents, and uses this syntax:

E[*type*] *address* [*list*]

CODEVIEW (CP)

The F (Fill) command sets memory contents to and iterated pattern, and uses this syntax:

F *range list*

The M (Move) command copies memory contents from one region to another, and uses this syntax:

M *range1 address2*

The A (Assemble) modifies a text range using a dynamic assembler. This is the syntax:

A [*address*]

## Execution control

The following commands control program execution:

| Execution Control | |
|---|---|
| g [*address*] | Resume execution (go). If provided, go to *address*. |
| t [*count*] | Trace; single-step; step into function on function call. |
| p [*count*] | Program; single-step; step over function calls (e.g. atomic). |
| e | "Slow motion" execute. |
| l [*args*] | Restart (load). Uses command line arguments if specified. |
| sig [*signo*] | Set pending signal; signal will take effect when execution is resumed with g, t or p. Without *signo*, display currently pending signal. |
| k | Stack trace. |

| Breakpoints | |
|---|---|
| bp [*address* [*passcount*] [*commands*]] | set a breakpoint |
| bc [*list* \| * ] | clear breakpoint(s) |
| bl | list breakpoints |
| be [*list* \| * ] | enable breakpoint(s) |
| bd [*list* \| * ] | disable breakpoint(s) |

CODEVIEW (CP)

| Watch, Watchpoints, & Tracepoints: | |
|---|---|
| w? *expression*[,*format*] | watch expression |
| wp? *expression*[,*format*] | set watchpoint |
| tp? *expression*[,*format*] | set tracepoint |
| w | watch, watchpoint and tracepoint list |
| y [*list* \| *] | watch, wathcpoint and tracepoint delete |

## Redirection Commands

| Re-direction commands: | |
|---|---|
| < *filename* | Re-direct codeview input from *filename*. |
| [T] > *filename* | Re-direct codeview output to *filename*. Optional argument **T** echos output to codeview screen as well as *filename*. |
| = *filename* | Re-direct input and output. |
| **comment* | Echo *comment* text to codeview screen. |
| : | Delay execution of commands by aproximately 1/2 second. |
| " | Pause until keystroke. |

## Miscellaneous commands

| Miscellaneous commands | |
|---|---|
| q | exit codeview |
| ![*shell command(s)*] | shell escape |
| @ | redraw codeview screen |
| h | help |
| <esc> | clear command line; cancel menu command |
| n[*radix*] | set default input/output radix (8, 10 or 16) with no argument, shows current default radix |
| <ctrl>g | "grow" current window, e.g. source or dialog |
| <ctrl>t | "shrink" current window, e.g. source or dialog |
| o[b\|c\|f][+\|-] | turn on or off or display current status of the following:<br>**b** - bytes coded<br>**c** - case sensitivity<br>**f** - flip/swap (e.g. save output) |
| #*number* | screen exchange; switch to output screen<br>tab set; specify width for tab to space expansion |

## Files

a.out
core
/usr/bin/cvtcoff

CODEVIEW (CP)

/usr/lib/cv.hlp
/usr/lib/keyboard/cv

## See Also

cc(CP), ld(CP), a.out(F), core(F), syms(F), sh(C), cvtcoff(CP), adb(CP), sdb(CP)

## Notes

Codeview executes transparently on either COFF or x.out binaries. The symbolic debug records from COFF binaries are translated to corresponding x.out format at startup using cvtcoff(CP). This converter must be installed for debugging of COFF binaries.

Only 386 binaries are supported.

In order to use <alt> keystrokes for menu selections and commands from the console, the keyboard mapping in */usr/lib/keyboard/cv* must be active. This can be accompilshed by linking the default mapping file to the codeview mapping:

**mv** */usr/lib/keyboard/keys* */usr/lib/keyboard/keys.00*
**ln** */usr/lib/keyboard/cv* */usr/lib/keyboard/keys*

Alternatively, you can use **mapkey** on the non-default file:

**mapkey** */usr/lib/keyboard/cv*

## Value Added

*Codeview* is an extension of AT&T System V provided by the Santa Cruz Operation.

VAS (K)

# vas: vasbind, vasmalloc, vasmapped, vasunbind

virtual address space memory routines

## Syntax

```
int
vasbind(paddr, vaddr, nbytes)
paddr_t paddr;
caddr_t vaddr;
unsigned int nbytes;

caddr_t
vasmalloc(paddr, nbytes)
paddr_t paddr;
unsigned int nbytes;

int
vasmapped(paddr, nbytes)
paddr_t paddr;
unsigned int nbytes;

int
vasunbind(vaddr, nbytes)
caddr_t vaddr;
unsigned int nbytes;
```

## Description

These routines allow a driver to map physical memory so that it can be read from or written to by both a driver and a calling user process. These routines are generally used to allow user processes to directly access video adapter memory. Memory that has been mapped using these routines is visible to the kernel and to a calling process. However, the mapping is not globally visible to all processes.

*vasmalloc* allocates virtual memory. Use this routine to obtain virtual address space that is not currently in use. *vasmalloc* currently can only allocate four megabytes of virtual address space on each call. Requests less than this amount are rounded up to four megabytes; requests larger than this amount cause an error to occur. The *nbytes* argument exists for forward compatibility. *vasmalloc* returns an address to virtual user memory; no actual physical memory is allocated by this routine.

VAS (K)

*vasbind* binds a specified virtual address to a physical address. This routine ensures that a problem will not occur with the bound memory being swapped out and causing a page fault and panic in the kernel. Before using *vasbind*, call *vasmapped* to determine if memory has already been mapped for the calling process.

The physical address supplied to *vasbind* may be the address of an I/O address space, for example, a memory-mapped I/O address. Or specify -1 to request that the memory be allocated from the kernel free memory pool.

When *vasbind* completes, the driver must pass the virtual address back to the user process using *copyout*(K) or another similar routine. Calls to *vasbind* must not specify an address in the text, data, or shared data segments of a user process.

The upper limit for user virtual memory is set in the KVBASE constant (defined in **sys/immu.h**); above KVBASE is the kernel virtual address space. The virtual address supplied to *vasbind* must be in user virtual memory (below KVBASE), and must not be in use by the current process.

*vasmapped* determines if a mapping is already in place.

*vasunbind* undoes a mapping.

## Notes

These routines cannot be called from a driver's interrupt routine (*xxintr*).

## Parameters

| | |
|---|---|
| *nbytes* | number of bytes of memory to allocate, bind, or unbind. |
| *paddr* | Physical address at which the specified virtual address is to be bound. When calling *vasbind*, *paddr* can be set to -1 to indicate that the requested user virtual memory is to be allocated. |
| *vaddr* | Virtual address to bind or unbind to or from physical memory |

## Return Value

*vasbind* returns -1 if an error occurs or if an error is found in

VAS (K)

**u.u_error**. *vasmalloc* returns a virtual address. *vasunbind* returns -1 if an error is found in **u.u_error**, or if the virtual address couldn't be found. *vasmapped* returns the virtual address at which the supplied physical address is bound, or 0 (zero) is physical address is not bound.

## See Also

sptalloc(K), copyout(K)

CLRBUF (K)

# clrbuf

zeros a block I/O buffer

## Syntax

```
#include "sys/buf.h"

int
clrbuf(bp)
struct buf *bp;
```

## Description

This routine zeroes a buffer previously allocated during block I/O. You supply a pointer to a buffer header (**bp**) to *clrbuf*. The *clrbuf* routine then zeroes the data pointed to by the address in **bp->b.un.b_words** for the number of bytes specified in **bp->b_bcount**. *clrbuf* also sets **bp->b_resid** to zero (0). *clrbuf* does not have a return value. The **b_words**, **b_bcount**, and **b_resid** fields are members of the *buf* structure and are defined in the **buf.h** header file.

Call the *geteblk*(K) routine before calling *clrbuf*. *geteblk* does not clear the buffer; *clrbuf* must be used to ensure that the buffer is cleared. The *geteblk* routine assigns a value to **b_bcount**. The standard way to call *clrbuf* is as follows:

```
bp = geteblk();      /* Get a buffer */
clrbuf(bp);          /* Clear it     */
```

## Warning

If your driver is using a non-standard buffering scheme managed with *buf* headers, ensure that **b_bcount** is set to a value the same or less than the size of the buffer. If **b_bcount** is set to a value larger than the current buffer size, data may be destroyed. **b_bcount** is set by *geteblk* to SBUFSIZE. SBUFSIZE is described in **/sys/fs/s5param.h** and varies by file system size.

## See Also

geteblk(K)

NMI (K)

# nmi

detects non-maskable interrupt

## Syntax

**extern int nmi_hook;**

## Description

The kernel provides the capability to detect and handle non-maskable interrupts (NMI) generated by a device. When a NMI is detected, a routine that you write that has previously been assigned to *nmi_hook* is automatically executed. The *nmi_hook* variable is typically assigned a routine name in the *xxinit* routine of a driver. Only one assignment to *nmi_hook* can be made for a computer. A typical use of a *nmi_hook* assignment is for a routine associated with an uninterrupted power supply (UPS).

NMIs are normally rare events generated by the motherboard or by device controllers when a parity error or other catastrophic error occurs.

## Note

When the routine set to *nmi_hook* is called, the kernel passes a pointer to the kernel stack as an argument to the routine.

## Warnings

The routine assigned to *nmi_hook* must first determine if the NMI was raised by the device associated with the device driver. Determine this status by examining a data area that is shared with the device to which the device would have previously written the status of the interrupt should it have raised it. In addition, it is the responsibility of the caller to store any previously set value from *nmi_hook* and to "pass it on" regardless if the NMI is responded to by your driver.

NMI (K)

## Example

This example depicts the following:

- How to store a previous value, line 6, stored in *previous_hook* and then pass it on by the driver's routine in line 18.
- Assigning a driver routine to *nmi_hook* in line 7. In the event that an NMI occurs, *bdm_nmi_hook* is called to determine if the interrupt was raised by the device associated with the driver.
- What a routine is like that checks a stored data area, *was_that_my_nmi*, line 12, to determine if the interrupt should be acted upon. If the interrupt should be handled, then the routine written by the driver developer, *service_nmi* is invoked in line 14.
- How to pass on a previous value set in *nmi_hook* and to keep the pointer to the kernel stack intact in the call, as described in line 18.

```
1   int (*previous_hook)();

2   bdminit()
3   {
4           extern int (*nmi_hook)();
5           extern int bdm_nmi_hook();

6           previous_hook = nmi_hook;
7           nmi_hook = bdm_nmi_hook;
8   }

9   bdm_nmi_hook(krnl_stack_ptr)
10  int krnl_stack_ptr;
11  {
12          if (was_that_my_nmi() == TRUE)
13          {
14                  service_nmi();
15                  return 1;
16          }

17          if (previous_hook)
18                  return (*previous_hook)(krnl_stack_ptr);
19          else
20                  return 0;
21  }
```

SIGSETV (S)

# sigset, sighold, sigpause, sigrelse, sigignore

System V signal management

## Syntax

```
#include <signal.h>

void (*sigset(sig, func))()
int sig;
void (*func)();

int sighold(sig)
int sig;

int sigpause(sig)
int sig;

int sigrelse(sig)
int sig;

int sigignore(sig)
int sig;
```

## Description

The functions **sigset**, **sighold**, **sigpause**, **sigrelse**, and **sigignore** enhance the signal facility and provide signal management for application processes.

The argument *sig* specifies the signal and the argument **func** specifies the choice. The argument *sig* can be assigned any one of the following signals except **SIGKILL**:

| | |
|---|---|
| SIGHUP | hangup |
| SIGINT | interrupt |
| SIGQUIT | quit* |
| SIGILL | illegal instruction (not reset when caught)* |
| SIGTRAP | trace trap (not reset when caught)* |
| SIGABRT | abort* |

| | |
|---|---|
| SIGFPE | floating point exception* |
| SIGKILL | kill (cannot be caught or ignored) |
| SIGSYS | bad argument to routine* |
| SIGPIPE | write on a pipe with no one to read it |
| SIGALRM | alarm clock |
| SIGTERM | software termination signal |
| SIGUSR1 | user-defined signal 1 |
| SIGUSR2 | user-defined signal 2 |

* The default action for these signals in an abnormal process termination. See **SIG_DFL**.

For portability, application programs should use or catch *only* the signals listed above; other signals are hardware- and implementation-dependent and may have very different meanings or results across systems. (For example, the System V signals **SIGEMT**, **SIGBUS**, **SIGEGV**, and **SIGIOT** are implementation-dependent and are not listed above.) Specific implementations may have other implementation-dependent signals.

The argument **func** is assigned one of four values: **SIG_DFL**, **SIG_IGN**, **SIG_HOLD** or an *address* of a signal-catching function. The argument **func** is declared as a type pointer to a function returning **void**. The following actions are prescribed by these values:

SIG_DFL Terminate upon receipt of a signal.

Upon receipt of the signal *sig*, the receiving process is to be terminated with all of the consequences outlined in **exit**(S). In addition, if *sig* is one of the signals marked with an asterisk above, implementation-dependent abnormal process termination routines, such as a core dump, may be invoked.

SIG_IGN Ignore signal.

Any pending signal *sig* is discarded. A pending signal is a signal that has occurred but for which no action has been taken. The system signal action is set to ignore future occurrences of this signal type.

SIG_HOLD Hold signal.

SIGSETV (S)

The signal *sig* is to be held. Any pending signal of this type remains held. Only one signal of each type is held.

Otherwise, *func* must be a pointer to a function. Upon receipt of the signal *sig*, the receiving process is to execute the signal-catching function pointed to by **func**. Any pending signal of this type is released. This address is retained across calls to other signal management functions, **sighold** and **sigrelse**. The signal number *sig* will be passed as the only argument to the signal-catching function. Before entering the signal-catching function, the value of **func** for the caught signal will be set to **SIG_HOLD**. During normal return from the signal-catching handler, the system signal action is restored to **func** and any held signal of this type is released. If a non-local goto is taken, the function **sigrelse** must be invoked to restore the system signal action and to release any signal of this type.

Upon return from the signal-catching function, the receiving process will resume execution at the point at which it was interrupted, except for implementation-defined signals where this may not be true.

When a signal to be caught occurs during a non-atomic operation such as a call to **read**(S), **write**(S), **open**(S), or **ioctl**(S) routine on a slow device (such as a terminal); or occurs during a **sigpause**(S) call; or occurs during a **wait**(S) routine that does not return immediately, the signal-catching function will be executed and then the interrupted routine may return a -1 to the calling process with **errno** set to **EINTR**.

The function **sigset** specifies the system signal action to be taken upon receipt of the argument *sig*.

The function **sighold** and the function **sigrelse** establish critical regions of code. A call to the function **sighold** is analogous to raising the priority level and deferring or holding a signal until the priority is lowered by the function **sigrelse**. The function **sigrelse** restores the system signal action to the action that was previously specified by the function **sigset**.

The function **sigignore** sets the action for the argument *sig* to **SIG_IGN**.

**sigpause** suspends the calling process until it receives a signal, like **pause**(S). However, if the signal *sig* had been received and held, it is released and the appropriate action is taken. This system call is generally used to test variables that changed when a signal is given. The correct strategy is to use **sighold** to block the signal and then test the variables. Then, if they have not changed, use **sigpause** to wait for the signal.

## Return Value

If successful, the function **sigset** will return the previous value of the system signal action for the specified signal *sig*; otherwise, it will

return **SIG_ERR** and **errno** will indicate the error.

For the functions **sighold**, **sigpause**, **sigrelse**, and **sigignore** a value of 0 will be returned upon success. Otherwise, a value of -1 will be returned and **errno** will indicate the error.

## Errors

Under the following conditions, the functions **sigset**, **sighold**, **sigpause**, **sigrelse**, and **sigignore** will fail and will set **errno** to:

| | |
|---|---|
| EINTR | A signal was caught during the execution of **sigpause**. |
| EINVAL | if *sig* is an illegal signal number or **SIGKILL** or if the default handling of *sig* cannot be changed. |

## Application Usage

For portability, application-programs should use only the symbolic names of signals rather than their values and use only the set of signals defined here. Specific implementations may have additional signals.

The other signal management routine, **signal**(S), should not be used in conjunction with these routines for a particular signal type.

## See Also

kill(S), pause(S), signal(S), wait(S), setjump(S)

CATGETS (S)

# catgets

read a program message

## Syntax

```
#include <nl_types.h>

char *catgets (catd, set_num, msg_num, s)
nl_catd catd;
int set_num, msg_num;
char *s;
```

## Description

*catgets* attempts to read message *msg_num*, in set *set_num*, from the message catalogue identified by *catd*. *catd* is a catalogue descriptor returned from an earlier call to *catopen (S)*. *s* points to a default message string which will be returned by *catgets* if it cannot retrieve the identified message.

## Return Value

If the identified message is retrieved successfully, *catgets* returns a pointer to an internal buffer area containing the null terminated message string. If the call is unsuccessful because the message catalogue identified by *catd* is not currently available, or if the message catalogue is available but the identified message is not contained therein, *s* (a pointer to the default message) is returned.

## See Also

catopen(S)

## Application Usage

The message-text is contained in an internal buffer area and should be copied by the application if it is to be saved or re-used after further calls to *catgets*.

CATOPEN (S)

# catopen, catclose

open/close a message catalogue

## Syntax

```
#include <nl_types.h>

nl_catd catopen (name, oflag)
char *name;
int oflag;

int catclose (catd)
nl_catd catd;
```

## Description

*catopen* opens a message catalogue and returns a catalogue descriptor. *name* specifies the name of the message catalogue to be opened. If *name* contains a "/" then *name* specifies a pathname for the message catalogue. Otherwise, the environment variable *NLSPATH* is used with *name* substituted for *%N* (see *environ*(M)). If *NLSPATH* does not exist in the environment, or if a message catalogue cannot be opened in any of the paths specified by *NLSPATH*, then */usr/lib/nls/msg/%l/%N.cat* is used.

*oflag* is reserved for future use and should be set to 0 (zero). The results of setting this field to any other value are undefined.

**Archives:**

If the special syntax for name:

<programname>@<archivename>

is used, then only archives are opened by catopen. The environment variable *NLSPATH* is used with the *archivename* part of name substituted for *%A* (see environ(M)). Only parts of the path containing a *%A* are recognized. When the archive is opened successfully, the entry named *programname* is used as message catalogue.

*catclose* closes the message catalogue identified by *catd*.

CATOPEN (S)

## Return Value

If successful, *catopen* returns a message catalogue descriptor for use on subsequent calls to *catgetmsg*, *catgets*, and *catclose*. If unsuccessful, *catopen* returns *(nl_catd)*-1.

*catclose* returns 0 if successful; otherwise it returns -1.

## See Also

catgets(S), environ(M)

## Standards Conformance

The capability to locate message catalogues in archives is specific to this implementation.

## Application Usage

Using *catopen* may cause another file descriptor to be allocated by the calling process, in addition to *stdin*, *stdout* and *stderr*. Applications should take care not to close this file descriptor by mistake.

DUMPMSG (CP)

# dumpmsg

generate a message source file

## Syntax

**dumpmsg** [-X] [-C] catfile msgfile

## Description

*dumpmsg* reads the formatted message catalogue *catfile* and generates a message source file *msgfile*. To read from standard input or write to standard output use '-' instead of the filename.

The following options are recognized:

-X this allows the use of *Sinix* extensions (for example, *preload* can be supported). *dumpmsg* normally complains about any format not defined in the X/Open portability guide.
-C non-ASCII codes are output as octal constants in the form *nnn*.

## Exit Status

*dumpmsg* returns 0 if successful. The following values are returned on error:

| | |
|---|---|
| **1** | fatal error occurred. |
| **2** | internal 'bug' occurred. |
| **10** | encountered warnings. |
| **20** | encountered errors (and maybe warnings). |

## Standard Conformance

This utility is specific to this implementation.

## See Also

gencat(CP), msginfo(F).

## Application Usage

A text editor may be used to make leading or trailing blanks visible.

The following shell script using **sed(C)** dumps the messages of message catalogue **chmod.cat**, adds the double-quote character (") to the dumped messages, surrounds all messages with the double-quote character ("), and copies the processed data to stdout:

DUMPMSG (CP)

```
dumpmsg -X chmod.cat - | { echo \$quote \" ; \
sed -e 's/^\([0-9][0-9]*[_\t]\)\(.*\)/\1"\2" /' ; }
```

In the above example, _ and \t represent the space character and the tab character, respectively.

GENCAT (CP)

# gencat

generate a formatted message catalogue

## Syntax

**gencat** [ **-X** ] catfile msgfile [ msgfile ... ]

## Description

*gencat* merges the message text source file(s) *msgfile* into a formatted message catalogue *catfile*. *catfile* will be created if it does not already exist. If *catfile* does already exist its messages will be included in the new *catfile*. If set and message numbers collide, the new message-text defined in *msgfile* will replace the old message text currently contained in *catfile*. *gencat* will complain about formats not defined in the **X/Open** portability guide.

The following options are recognized by *gencat:*

| | |
|---|---|
| - | use standard output or standard input in the place of (respectively) *catfile* or *msgfile*. |
| -X | allows for usage of *Sinix* extensions such as selection of different load modes and/or recognition of extra keywords. |

## Exit Status

*gencat* returns 0 on success or 1 on failure.

## Standards Conformance

The **-X** option is specific to this implementation.

## See Also

msginfo(F).

## Application Usage

Message catalogues produced by *gencat* are binary encoded, meaning that their portability cannot be guaranteed between different types of machines. Therefore, just as C programs need to be recompiled for each type of machine, message catalogues must be recreated via *gencat*.

ICONV (CP)

# iconv

international codeset conversion

## Syntax

**iconv** [**-h**] [**-d**] *-f fromcodeset -t tocodeset* [*file...*]

## Description

The *iconv* command converts the encoding of characters in *file* from *fromcodeset* to *tocodeset* and writes the results on the standard output. For example:

*iconv -f IS8859 -t IS6937 datafile*

converts the file *datafile* from IS8859 encoding to IS6937 encoding, reads the conversion tables from *<path-prefix>/conv/IS8859_IS6937*, and writes the results on the standard output.

The conversion tables are held in *fromcodeset_tocodeset* in subdirectory *conv* under ICONV which is by default **/usr/lib/nls**. Each line in the conversion table specifies a conversion from the first string to the second string on the same line. Spaces and tabs are used to delimit tokens, the \ character is used to escape the next character and to introduce (exactly) three digit octal constants. Blank lines and lines starting with a hash *(#)* are introduced.

```
#   a sample conversion table
#   fromcode tocode

    \#          wasahash
    \040        wasablank
    string      another\040string
```

The options are:

**-h** Prints a usage message on the standard error output.

**-d** Deletes any characters which are not listed in the conversion table (that is, invalid characters). The default is to pass through any invalid characters.

## Notes

As the name of the file holding the conversion rules is made with the characters of *fromcodeset*, *tocodeset* and the character _, there may be truncation some significant characters of the file name in order to

ICONV (CP)

conform to the maximal file name length {NAME_MAX}.

## Files

$ICONV/conv/fromcoset_tocodest

MAR (CP)

# mar

## message catalogue archive and library maintainer

## Syntax

**mar** key [ option ] afile name ...

## Description

*mar* maintains groups of message catalogue files created by *gencat*(CP) combined into a single archive file. Its main use is to create and update library files as used throughout the message catalogue system. Any leading path will be stripped when referencing files that are already part of the archive, hence

mar d msgarchive /usr/local/msgfile1

is equivalent to

mar d msgarchive msgfile1

and will delete the file named *msgfile1* from the archive named *msgarchive*.

*key* is one character from the set **drtpx,** optionally concatenated with **v.** *afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

**d** Delete the named file(s) from the archive file. If no names are given, all files in the archive will be deleted.

**r** Replace the named files in the archive file or append them to the archive file, if they are not already part of the archive. The archive file will be created if it does not already exist.

**t** Print a table of contents of the archive file. If no names are given, all files in the archive are printed. If names are given, only those files named will be printed.

**p** Print the named files to stdout. If no names are given, all files will be printed. When piped to *dumpmsg*(CP) as

mar p msgarchive msgfile1 | dumpmsg - -

a listing of all the messages in the archive will be printed on stdout.

**x** Extract the named files. If no names are given, all files in the archive will be extracted. Neither the **x** option nor the **p** option will alter the archive file.

**v** Verbose. Under the verbose option, *dxtr* give a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t,** it gives a long listing of all information about the files, including the name of the file, its last update, its size and the name of the message catalogue as referenced by *catopen*(S), *catgets*(S), and *catgetmsg*(S). The size of

MAR (CP)

the file is identical to the size as reported by *ls*(C).

## Files

/tmp/msga msga temporaries

## Environment Variables

**LC_COLLATE** affects the order in which the output is sorted for the **-t** option; and **LC_TIME** affects the contents of date and time strings output by the **-t** and **-v** options. **LANG** provides the necessary defaults if any of these variables are not defined in the environment. If **{LANG}** is not defined, output is sorted in machine collation order, date and time strings default to system specific formats.

## Standards Conformance

This utility is specific to this implementation.

## See Also

gencat(CP), dumpmsg(CP), ar(CP), catopen(S)

## Application Usage

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

*mar* will be rather slow on big archives even when only deleting a single file.

## Note

This version of *mar* uses a binary format archive which is not portable among the various machines running UNIX. Thus, just as C programs have to be recompiled for each type of machine, message catalogue archives have to be recreated by *mar* as well.